BLACK BOX MODELING OF HYDROCARBON PHYSIOCHEMICAL PROPERTIES AND ITS APPROACH TO FUEL OPTIMIZATION

A Thesis

by

ANDREW YUEH

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Mahmoud M. El-Halwagi
Co-Chair of Committee,	Nimir O. Elbashir
Committee Members,	M. Sam Mannan
	Sergiy Butenko
Head of Department,	M. Nazmul Karim

December 2015

Major Subject: Chemical Engineering

ABSTRACT

An energy pathway of great interest is gas-to-liquid (GTL) technologies, which converts natural gas to valuable chemicals and fuels. Three powerful regression methods were implemented, to create models for accurate predictions of physical properties– density, freezing point, flash point, and heat content. With the use of experimental training data, three distinct techniques were performed and analyzed: artificial neural networks, support vector machine (SVM) and Kriging modeling.

For further accuracy, optimal simulation settings were elucidated through repeated runs and rigorous testing, with substantial increases in performance in low performing models. Most models generated were accurate with good trends, except freezing point. A formulation package called DataMine, coded in R, was created for current work and future endeavors.

DEDICATION

To

My family, who have been supportive of all of my academic endeavors.

ACKNOWLEDGEMENTS

My deepest gratitude goes toward Dr. Mahmoud El-Halwagi, Dr. Nimir Elbashir, Dr. M. M. Faruque Hasan, and Dr. Ashraf Ibrahim. Their guidance has led me through this learning experience– culminating to the findings of my academic endeavors. Special thanks go toward my committee members, Dr. Sam Mannan and Dr. Sergiy Butenko for their participation and assessment of my work.

I would also like to thank fellow scholars: Dr. Debalina Sengupta, Rendra Bayu, and Nolan Worstell. Their influence and support, both active and passive, have driven me to the conclusion of my efforts here, in the Artie McFerrin Department of Chemical Engineering, of Texas A&M University.

This research was made possible by NPRP grant no. 5-066-2-023 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER I
INTRODUCTION

I.1 Background

As global economies grow and emerging markets are formed, travel increases. The aviation industry expects air travel to double over the next 20 years. (Pearce 2015) Parallel to this, natural and shale gas utilization has become a booming industry to fulfill modern energy needs. One pathway of great interest is gas-to-liquid (GTL) technologies, which is aimed at converting natural gas (methane) to chemicals and ultra-clean fuels, via Fischer Tropsch chemistry. The general schematic and process behind the conversion, shown in Figure 1, of natural gas to liquid fuels is well established: Steam reforming, Fischer Tropsch, and a hydrocarbon distillation column (Opec 2013).

$$CH_4 + H_2O \rightarrow H_2 + CO_2 \quad (1.1)$$

$$H_2O + CO \rightarrow H_2 + CO_2 \quad (1.2)$$

$$(2n + 1)H_2 + nCO \rightarrow C_nH_{(2n+2)} + nH_2O \quad (1.3)$$

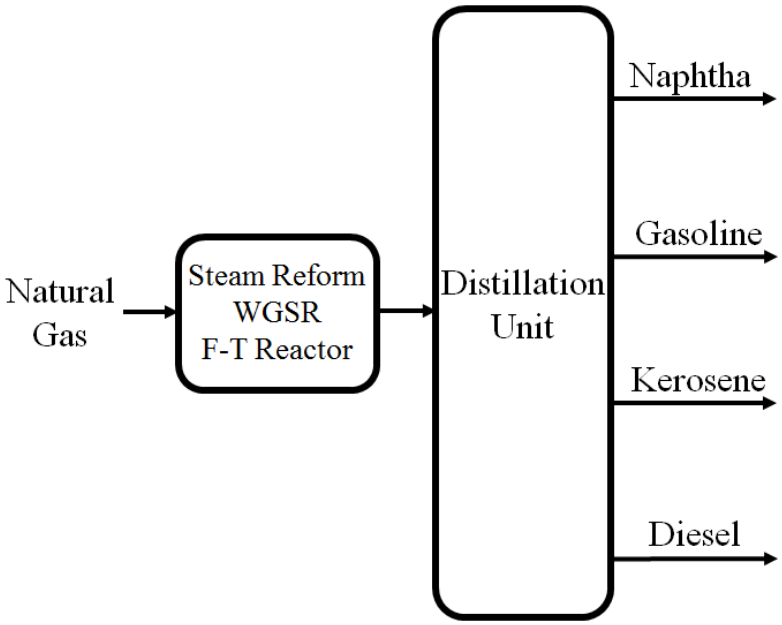

Figure 1 Basic GTL scheme and the liquid hydrocarbons derived from the process.

Gas-to-liquid, coal-to-liquid (CTL) and biomass-to-liquid (BTL) are particularly attractive from an environmental standpoint. Global warming regulations continually cast its influence in the energy industry. GTL-based fuels have virtually no sulfur or aromatic hydrocarbon content, classifying GTL, CTL and BTL fuels as ultra-clean fuels, when compared to crude oil based fuels. (Al-Nuaimi, et al. 2014) The implementation of these processes and the sheer volume of this valuable resource has spurred this energy revolution, as shown in Figure 2.

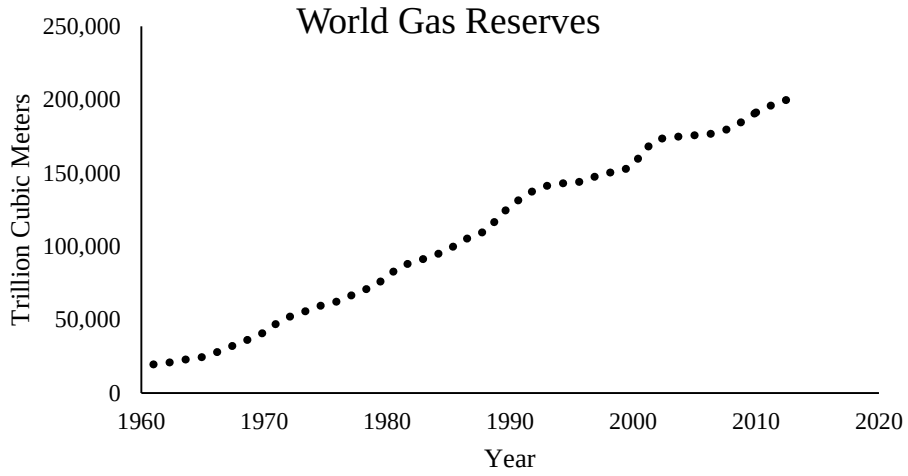

Figure 2 Global discovery of proven natural gas reserves, by volume. *(Opec 2013)*

Synthetic fuel cuts from GTL are similar to crude oil fuels, in terms of hydrocarbon distributions. These hydrocarbon composition disparities, however, create differences in physical properties, when compared to conventional jet fuel. (Al-Nuaimi, et al. 2014) It is important to observe and account for the differences between these fuels, for their safety and implementation.

Significant research efforts have been conducted at Texas A&M University Qatar campus (TAMU-Q) in the field of GTL derived synthetic fuels, including jet fuels. Valuable data on production and characterization are being developed and assessed experimentally in Prof. Elbashir's labs at TAMU-Q. (Al-Nuaimi, et al. 2014)

A key need for large-scale utilization of GTL jet fuel is the ability to ensure compliance with fuel standards. Currently, GTL jet fuel does not meet physical property standards set by the American Society for Testing and Materials (ASTM) D-1655, in Figure 3. (Al-Nuaimi, et al. 2014)

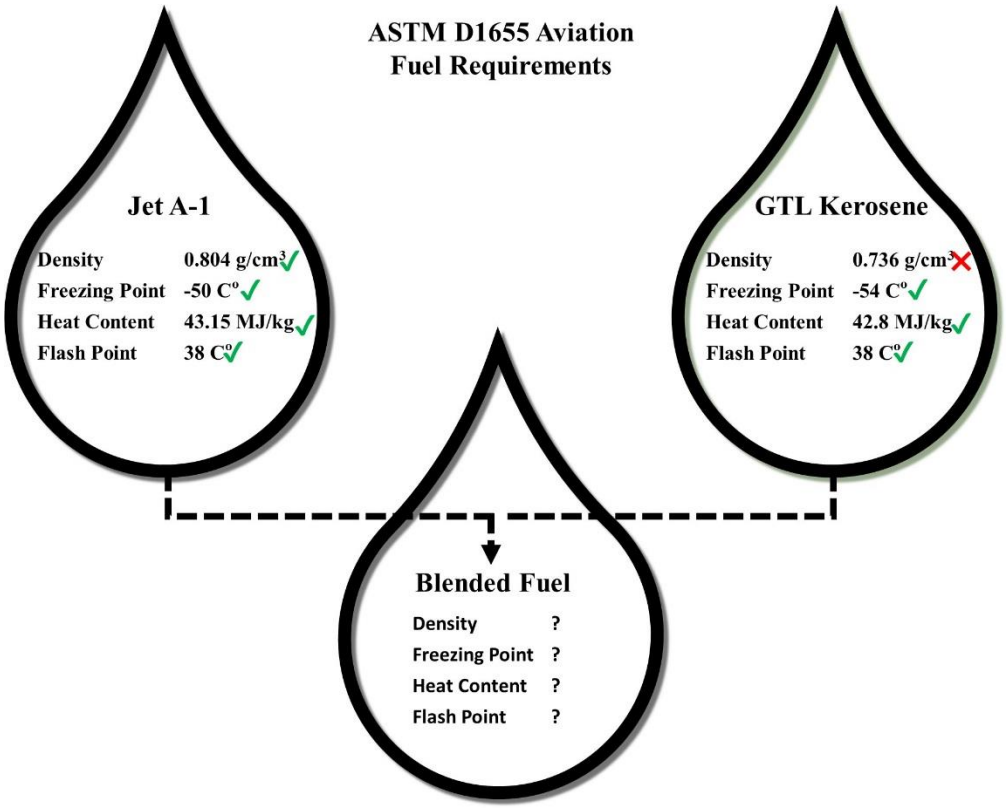

Figure 3 Visualization of the conceptual goal of fuel blending for aviation fuel optimization.

A recent standard, however, has allowed a 50%-50 blend of GTL derived synthetic jet fuel and Jet A-1 (ASTM D-7566). (Bohra, et al. 2014) This is conceptually shown in Figure 3. The past decades have seen massive increases in proven natural gas reserves. This new standard is of capital interest for the future of synthetic fuels– it is one step toward large scale utilization of this abundant resource.

Recent work by Al-Nuaimi et al., (2014) has considered the role of cyclo-paraffins, n-paraffins, iso-paraffins and the addition of small amounts of mono aromatics to synthetic jet fuel blends. (Al-Nuaimi, et al. 2014) The design of new generations of synthetic fuels, while exploring their impact on global jet fuel markets, requires a thorough understanding of the relationship between a fuel's fundamental hydrocarbon composition and the physical properties it exhibits.

I.2 Objectives

This research seeks to build upon prior work by Al-Nuaimi et al., (2014) and formulate new and novel methods to address the same aims: generating an algorithmic process that accurately predicts physical properties of hydrocarbon solutions. This is based on experimental data whose physical properties are already known. Using this data, operations were performed, as follows.

1. Formulate three advanced regressions algorithms; Kriging interpolation, artificial neural networks, and support vector machine (SVM). The goal is creating predictive models for the physical properties of density, freezing point, flash point, and net heat of combustion.
2. Further establish and implement methods for increasing accuracy in predicted values, through rigorous testing and elucidation of optimal parameters.

CHAPTER II
LITERATURE REVIEW

II.1 Optimization of Jet Fuel Blends through the Analysis of its Physical Properties

Al-Nuaimi et al. (2014) explores the statistical correlations of fuel properties, based on hydrocarbon composition. These models attempt to mimic the properties of various fuels due to their properties, shown in Table 1. The work omits an aromatic hydrocarbon, the fourth fundamental hydrocarbon.

Table 1 Definition and properties of each respective hydrocarbon mixture.

Hydrocarbon source	Definition and properties
Jet A-1 fuel	• Petroleum derived jet fuel– the conventional source • Compliant with ASTM D-1655 standards
Synthetic paraffinic kerosene (SPK)	• Gas-to-liquid derived jet fuel, an unconventional fuel • Predominantly composed of cyclo-paraffins, n-paraffins, and iso-paraffins • Is aromatic deficient • Hydrocarbon predominately range between C_7 to C_{16} • Incompliant with ASTM D-1655 standards
Shell SolT™ (Sol T)	• Industrially Purchased iso-paraffins • Iso-paraffin range of C_{11} to C_{12}

Two distinct methodologies were applied:

The first experiment was the binary blending of Jet A-1 fuel and SPK. Nine iterations were performed, with 10% incremental increases/decreases for each respective fuel. This is a permutation of the established ASTM D-1655 standard (50:50 blends of Jet A-1 and SPK). (Al-Nuaimi, et al. 2014) Physical properties of each of the nine blends were measured discussed.

The second set of experiments was composed of varying compositions of distinct and fully quantified hydrocarbons. Hydrocarbon classes of n-paraffins, cyclo-paraffins, and iso-paraffins were measured and mixed, with full knowledge of the exact composition of each solution. Molecular constituents used were decane, decaline, and Sol T, respectively.

Al-Nuaimi et al. (2014) used the experimental data for regression analysis of the four physical properties: density, net heat of combustion, flash point, and freezing point. Results are as follows. Polynomial equations were elucidated for this modeling.

A linear regression model for density was effective, shown in Figure 4. For net heat of combustion, in Figure 5, a well correlated linear model was realized.

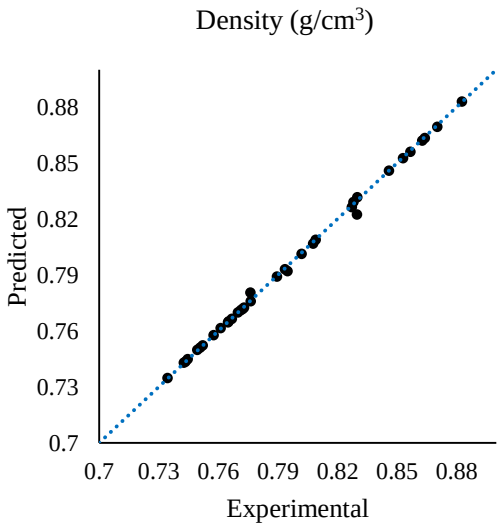

Figure 4 Experimental data versus model predictions for density (Al-Nuami et al., 2014).

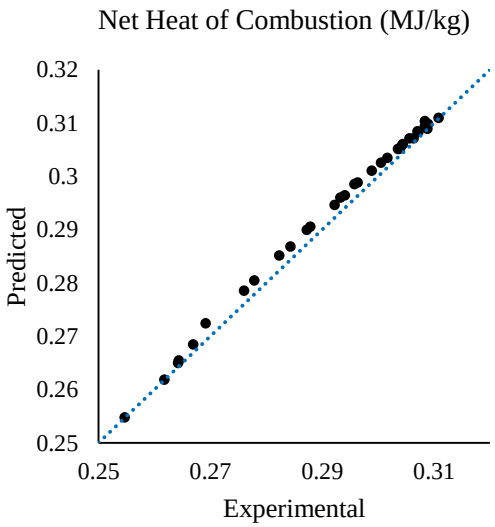

Figure 5 Experimental data versus model predictions net heat of combustion (Al-Nuami et al., 2014).

Flash point and freezing point in Figure 6 and 7, respectively, were not as conclusive. The correlations are nonlinear, making modeling difficult. Al-Nuami et al. (2014) proposed nonlinear models for each respective physical property, but the results

are far less conclusive than density and net heat of combustion.

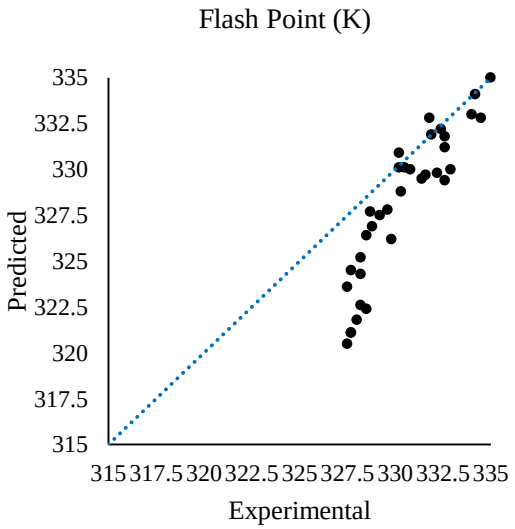

Figure 6 Experimental data versus model predictions for flash point (Al-Nuami et al., 2014).

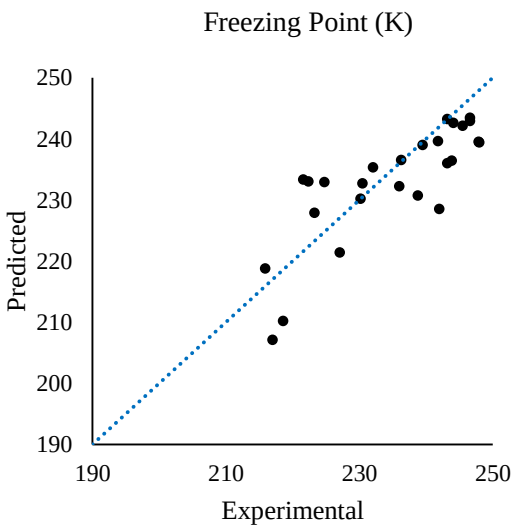

Figure 7 Experimental data versus model predictions for freezing point (Al-Nuami et al., 2014).

II.2 Role of Aromatics in Synthetic Jet Fuels

A previous study of Prof. Elbashir's team Bohra, Selam, & Hafis, 2007 report provides insight on the effect of aromatic hydrocarbons on synthetic fuels. It quantifies and analyzes fuel systems and aromatic influence on its physical properties. This was especially helpful for the current; the findings in this report allow initial assessments on the nature of the data that are provided and analyzed throughout this research.

Experimentally, the physical properties of interest that were analyzed were: density, flash point, freezing point, and net heat content. These physical properties are part of the criteria that defines the jet fuel ASTM standard. The aromatic compound used in this study was styrene. The synthetic paraffinic kerosene used was the Pearl SPK– obtained from Shell's Pearl GTL plant in Qatar. (Bohra, et al. 2014)

Density is the key property that prevents pure Pearl SPK from ASTM compliancy. (Al-Nuaimi, et al. 2014) Density of SPK is increased with increasing

aromatic content, as expected and shown in Figure 8. Freezing point is suppressed as aromatic composition of the fuel increases shown in Figure 9. (Bohra, et al. 2014)

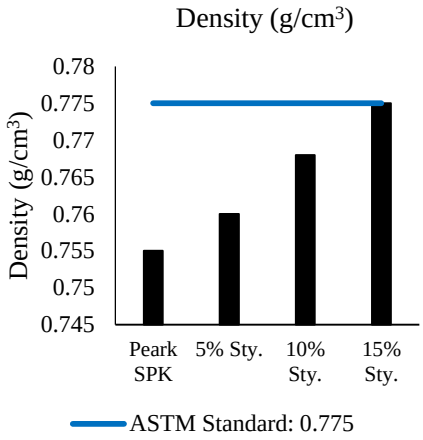

Figure 8 Density changes with respect to various styrene compositions. The blue bar is the minimum ASTM density. (Bohra, et al. 2014)

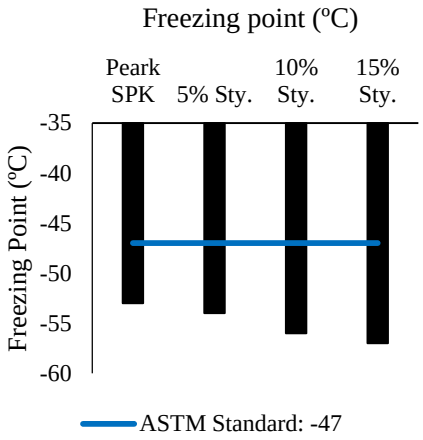

Figure 9 Freezing point changes with respect to various styrene compositions. The blue bar is the maximum ASTM temperature. (Bohra, et al. 2014)

Flash point is heavily suppressed; it presents itself as a potential detriment to physical viability of certain blends, shown in Figure 10. Heat of combustion is also suppressed with increasing aromatic content in the SPK, shown in Figure 11. (Bohra, et al. 2014)

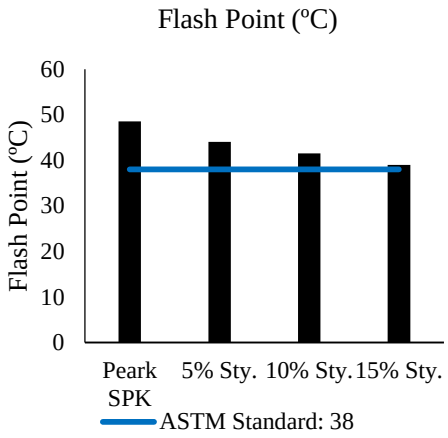

Figure 10 Flash point changes with respect to various styrene compositions. The blue bar is the minimum ASTM density. (Bohra, et al. 2014)

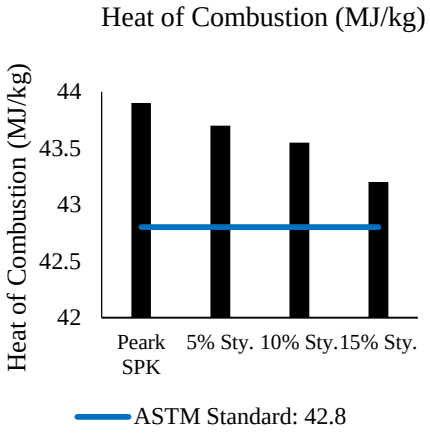

Figure 11 Heat of combustion changes with respect to various styrene compositions. The blue bar is the maximum ASTM temperature. (Bohra, et al. 2014)

CHAPTER III
METHODOLOGY AND APPLICATIONS

III.1 Conceptual Methods

The utilization of predictive model all shares the same general methods. Datasets were provided for density, freezing point, flash point and heat content, with the number of points shown in Table 2, and the numerical values in Appendix B.

Table 2 The amount of data for each respective physical property, used to create predictive models.

Physical Property	Datasets
Density	32
Freezing Point	32
Flash Point	13
Heat Content	16

With the data set provided, the following algorithmic system was applied. This process, in Figure 12, is repeated in a cross referencing and Monte Carlo optimization scheme.

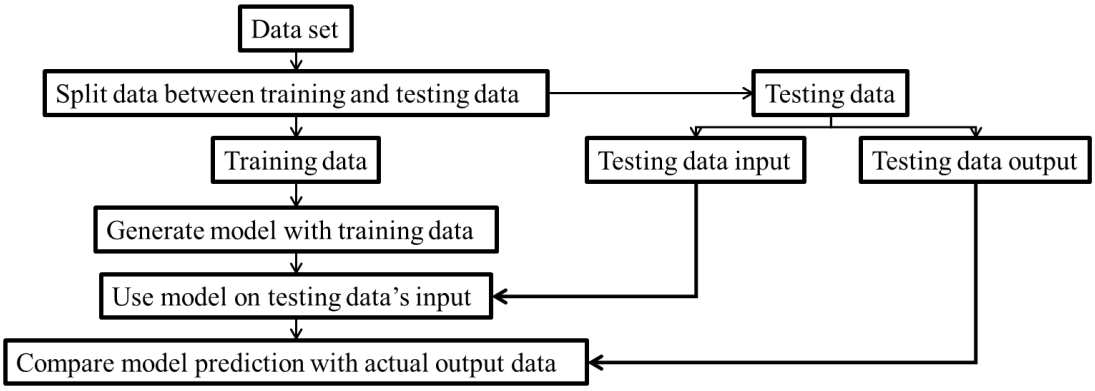

Figure 12 The systematic optimization method to accomplish objectives of this thesis.

III.2 R, the Programming Language and the Functions Coded

Using a powerful statistical programming language, R, a formulaic and comprehensive function was created, named DataMine. This function is the coded methodology in in Figure 12. DataMine is a single function with a full wrapper to handle every algorithm used in this research endeavor: Kriging interpolation, artificial neural networks, and SVM. All functions are outlined in Table 3. Some of the user inputs are specific to their respective algorithm. These parameters will be further divulged in the sections following this section.

Table 3 List of functions contained in the package DataMine

errorplot	Error Plotting
fuelComp	Hydrocarbon Distribution from the Blending of Distinct Fuels
krigcomp	Kriging Interpolation Compute
krigOpt	Kriging Interoplation Optimization: Predictions of Physical Properties of Theoretical Hydrocarbon Blends
krigSim	Kriging Optimization
neuralcomp	Artificial Neural Network Compute
neuralOpt	Artificial Neural Network Optimization: Predictions of Physical Properties of Theoretical Hydrocarbon Blends
neuralSim	Artificial Neural Network Optimization

Table 3 Continued.

scatterplot	Scatter Plotting
svmcomp	Support Vector Machine Compute
svmOpt	Support Vector Machine: Predictions of Physical Properties of Hydrocarbon Blends
svmSim	Support Vector Machine (SVM) Optimization

CHAPTER IV
REGRESSION ANALYSIS: ARTIFICIAL NEURAL NETWORKS

IV.1 Artificial Neural Networks: Introduction

Data mining has proved itself to be one of the most powerful regression methods for the prediction for physical phenomena in any field of study. Artificial neural networks, or ANN, are one of the most widely used methods of data mining. For this algorithm, the relationship between input and output does not need to be predefined for its effective use. (Hoak 2010) It is able to model highly nonlinear systems with more confidence than traditional regression methods.

Its versatility lies in the algorithm's flexibility in calculation parameters and methods. That being said, it should be used with an element of caution, as user settings may generate models that underfit, overfit, or be generally inaccurate. (Aslett 2015) In R, the package neuralnet was used to create neural networks for each physical property. (Fritsch, Guenther and Suling 2012) Within the package, the parameters that are carefully observed are the optimization algorithm; the number of hidden nodes; and the optimal amount of training data to use.

IV.2 Artificial Neural Networks: Training Data Optimization

The amount of data used to train a neural network is an impactful decision. There needs to be a balance in the number of data points for training and testing. A high training count can leave too few samples to confidently test, or validate, the model; while a low training count can lead to poor predictions. (Fritsch, Guenther and Suling 2012) The nature of data set cannot be assumed to be the same; data point counts differ; trends are unique; and the accuracy of instrumentation varies. With the importance of this concept and intrinsic variability of each chemical property, each physical property was individually tested.

Density trends are linear in character. 32 sample points were used as training data for its model, shown in Figure 13. Freezing point, in relative terms, benefits well when

more training data is used, shown in Figure 14. Based on the nature of improvement, it can be concluded, with relative confidence, that this physical property exhibits nonlinear trends. 32 samples were used to model and 4 samples to test was the best option.

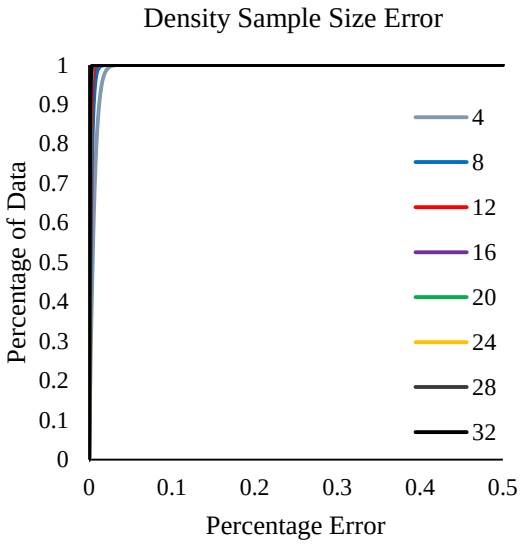

Figure 13 Density error plot for various sample sizes.

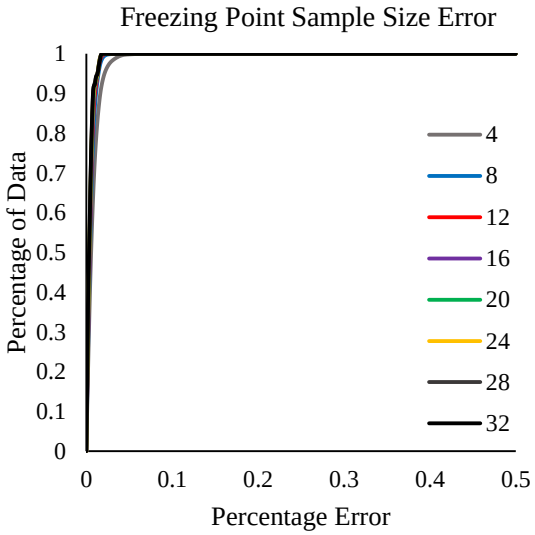

Figure 14 Freezing point error plot for various sample sizes.

For flash point, a model generated from 12 sample points creates erratic predictions. As visualized in Figure 15, sample size of 8 yields more consistent results; a counterintuitive conclusion. The utilization of an artificial neural network requires more data with increasingly complex systems. Typically, neural networks handle white noise well. (Fritsch, Guenther and Suling 2012) But in this circumstance issues such as; a nonlinear system; potential over fitting; and a poor number of data points are all factors, which contribute to the decision of using a sample size of 8.

As expected, heat content's error drops as sample size increase as shown in Figure 16; albeit not to the same efficacy nor magnitude as density and freezing point. 12 samples were used to train the heat content neural network.

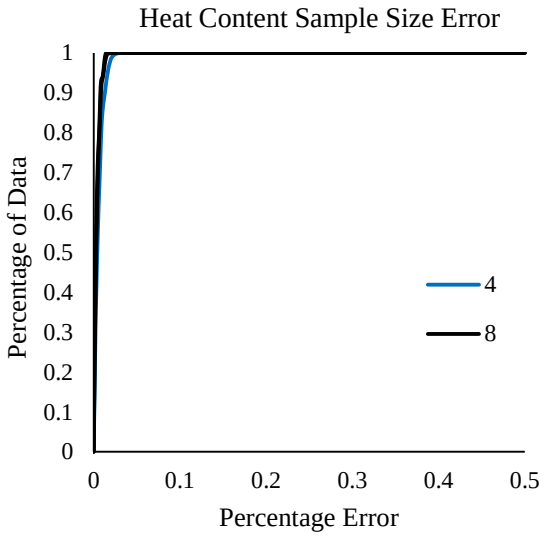

Figure 15 Heat content error plot for various sample sizes.

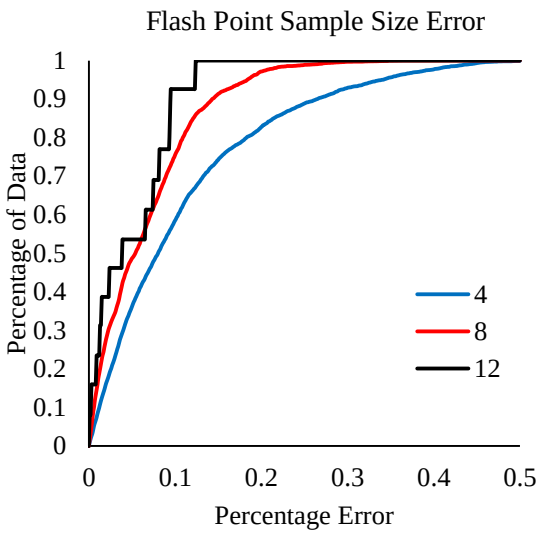

Figure 16 Flash point error plot for various sample sizes. Issues with nonlinearity generates erratic results with fewer testing points.

As sample size increases, error drops. Freezing point and density converge to their lowest RMSE possible, as sample size increases. Flash point and heat content, however, show a distinct limitation in error reduction.

IV.3 Artificial Neural Networks: Hidden Nodes

Aside from training size, another parameter to be mindful of is the number of hidden nodes that are utilized in a neural network. Hidden nodes determine the number of intermediate equations and calculations that eventually lead to the desired output. (Fritsch, Guenther and Suling 2012) An example is shown in Figure 17.

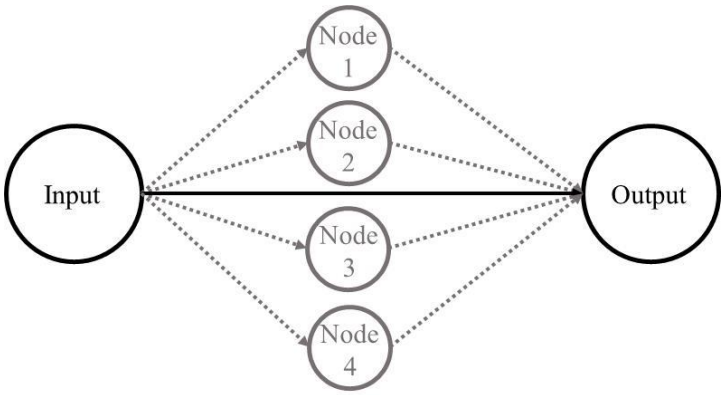

Figure 17 A sample neural network. In black, the visible concept of input and output generation in a neural network. In gray is the actual calculation pathway with 4 hidden nodes.

Each physical property was modeled using nodes of various counts; models ranging from 1 to 7 nodes. Conceptually, the number of hidden layers affects accuracy; typically, compromises must be accurate but not over fit. (Aslett 2015)

For density, the number of nodes used in a model proved to be trivial. This was an expected result, due to the linear nature of the physical property, shown in Figure 18. The lowest RMSE was with 1 hidden node.

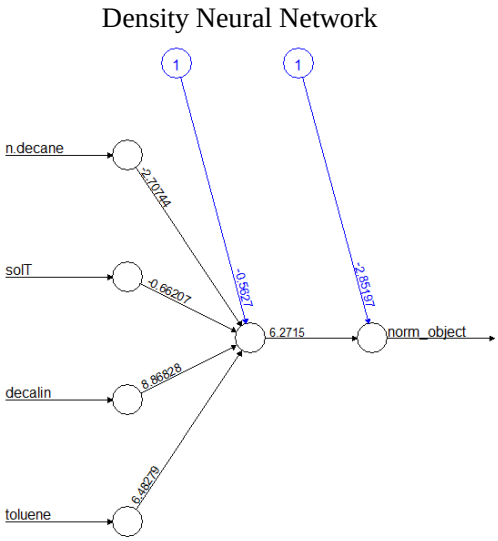

Figure 18 Visual model of a single node density neural network.

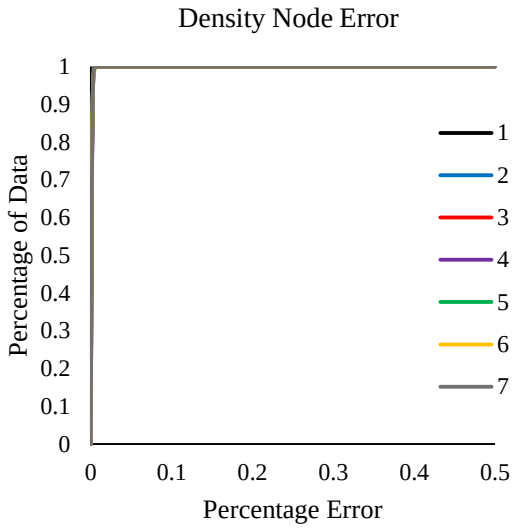

Figure 19 Error plot of models with various node counts.

Freezing point node appears to be trivial, as shown in Figure 19. While the numeric comparison between the respective nodes is small, any level of error reduction impacts this nonlinear model. This is apparent when the model is applied to create predictions. For freezing point, the lowest RMSE was with 4 hidden nodes, visualized in Figure 18.

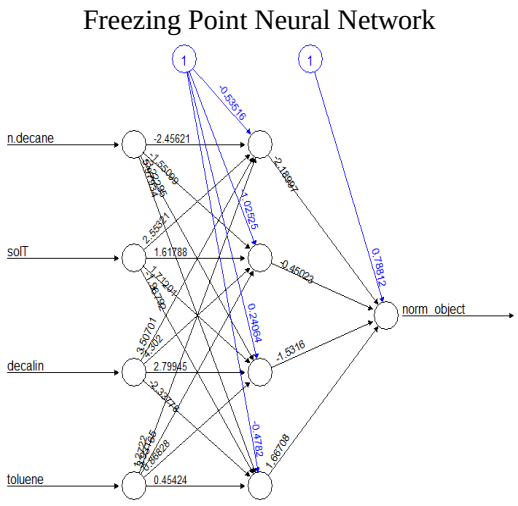

Figure 20 Visual model of a 4 node freezing point neural network.

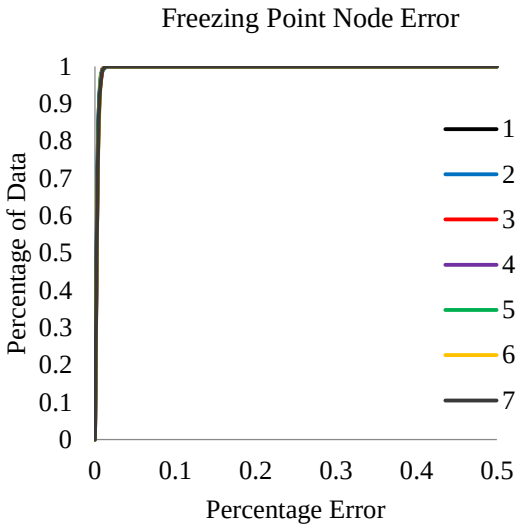

Figure 21 Error plot of models with various node counts.

Heat Content is sensitive to the number of nodes used, as shown in Figure 23. Flash point modeling is intrinsically nonlinear; care is used when deciding the number of nodes. It was found, through simulations that the lowest RMSE was with 5 hidden nodes, shown in Figure 22.

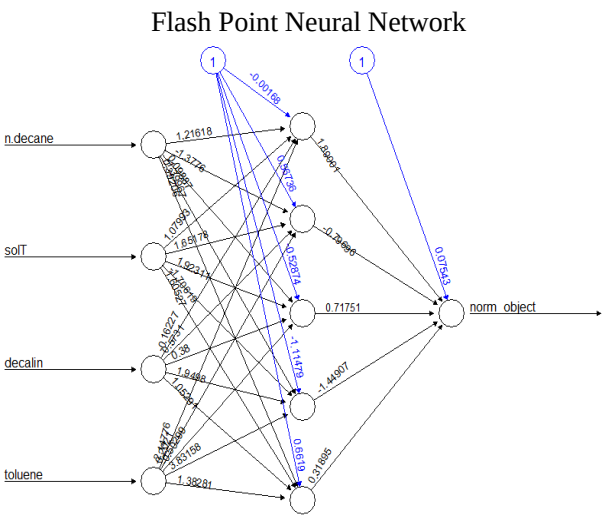

Figure 22 Visual model of a 5 node freezing point neural network.

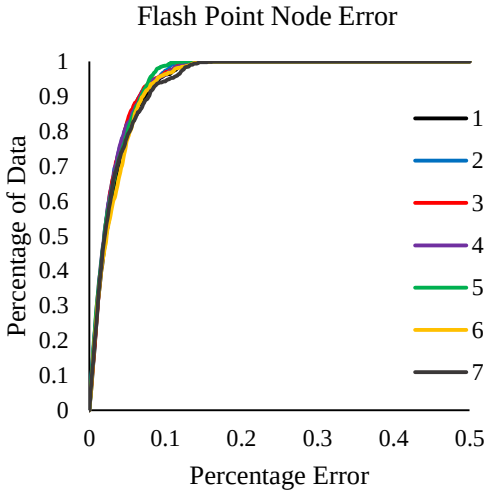

Figure 23 Error plot of models with various node counts.

IV.4 Artificial Neural Networks: Monte Carlo Simulations

With the best training size and number of hidden nodes established, shown in Table 4, important parameters were systematically established; subsequent cross referenced simulations were performed.

Table 4 Optimal hidden nodes and sample sizes used for neuralnet, the optimization function.

Physical Property	Hidden Nodes	Sample Size
Density	3	32
Freezing Point	6	32
Heat Content	14	12
Flash Point	14	8

The goal was to find the best internal weight parameter. This parameter is highly complex. It represents the values each neuron takes on. This creates a large number of weights; an upwards of 32 values. This parameter is far too complex to optimize systematically. Because of this, a Monte Carlo optimization scheme was utilized, shown in Figure 24.

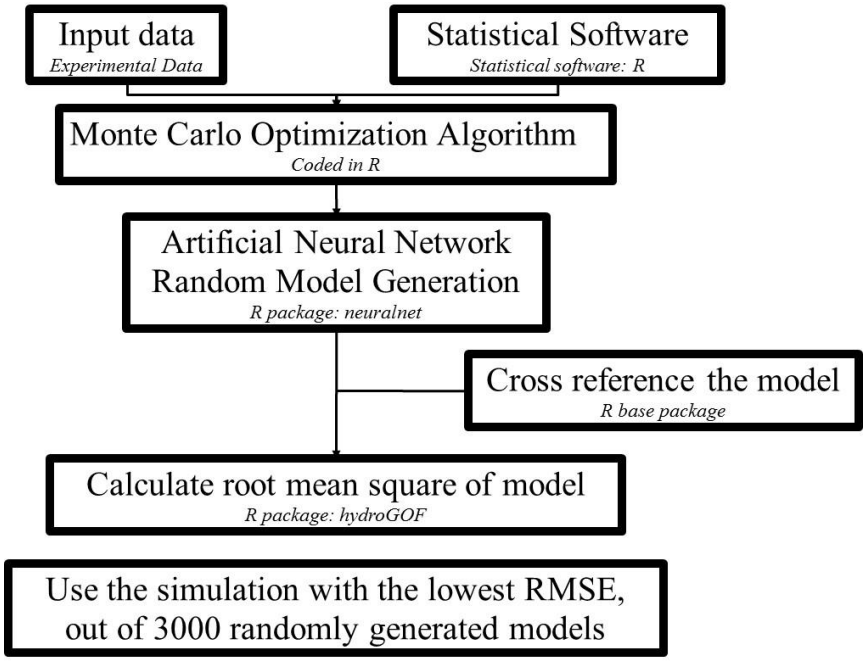

Figure 24 Flow chart of Monte Carlo optimization scheme, coded in R.

Artificial neural network models are stochastically generated in the neuralnet, in R. (Fritsch, Guenther and Suling 2012) Each model was cross referenced 500 times, yielding 12,000 generated values. The RMSE of each weight's experimental vs. predicted values were compared.

The relative error between the optimized weights and non-optimized weights ranged from trivial to substantial. Flash point outputs gain a substantial amount of confidence. Density, freezing point, and heat content show improvements, albeit with

lower improvement. These physical properties have either larger data sets or have more linear character; they stand to gain less from this form of optimization.

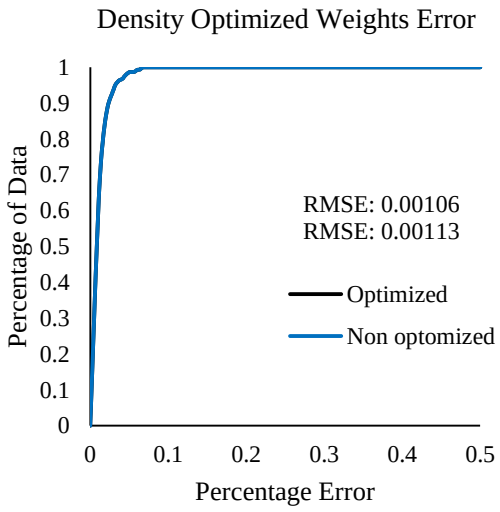

Figure 25 Density error plot visualization of the impact from the optimized algorithm.

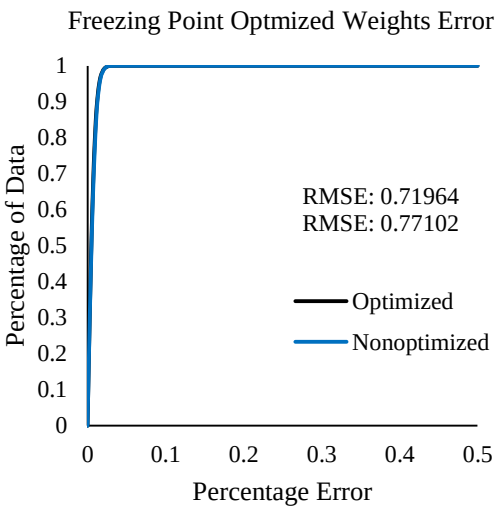

Figure 26 Freezing point error plot visualizatio of the impact from the optimized algorithm.

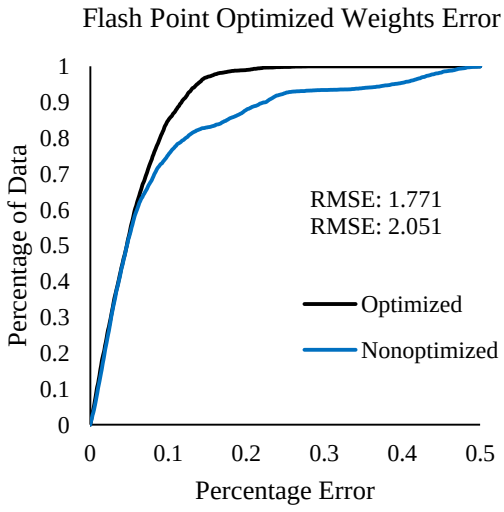

Figure 27 Flash point error plot visualization of the impact from the optimized algorithm.

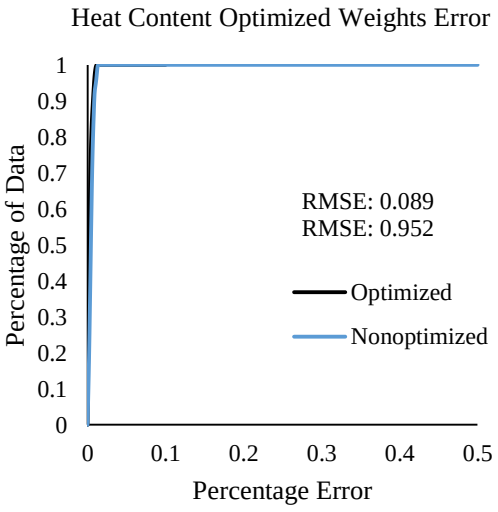

Figure 28 Heat content error plot visualization of the impact from the optimized algorithm.

IV.5 Artificial Neural Networks: Results

Artificial neural network optimization was accomplished using R, the open source programming language. The function neuralnet was used. Optimal parameters were found for training size and number of hidden nodes. With Monte Carlo optimization, the technique utilized, optimal predictions were found.

Much like the work done by Al-Nuaimi et al., the culmination of these optimization techniques results in the direct comparison of the model predicts compared to experimental values. (Al-Nuaimi, et al. 2014) A neural network approach to this objective is what differentiates this work from techniques described in Al-Nuaimi et al, 2014.

It should be noted that density, the physical property of interest, is accurately quantified, shown in Figure 29. Given the objective of this research, this is highly successful. However, the rest of the physical properties suffer in accuracy; freezing point, heat content, and flash point, as shown in Figure 31, Figure 32 and Figure 33, respectively.

The following scatterplots cumulatively represent the results of these optimized artificial neural network algorithms. Each predicted data points are an average of ~2800 cross referenced values. Numerical weights utilized and values represented in each scatterplot may be found in Appendix C.

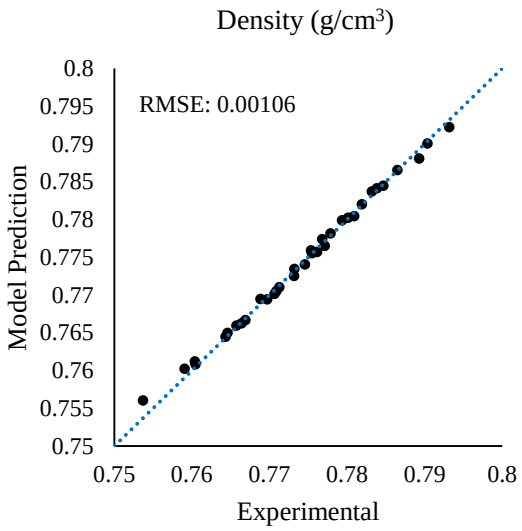

Figure 29 Density neural network comparison of experimental and model predictions.

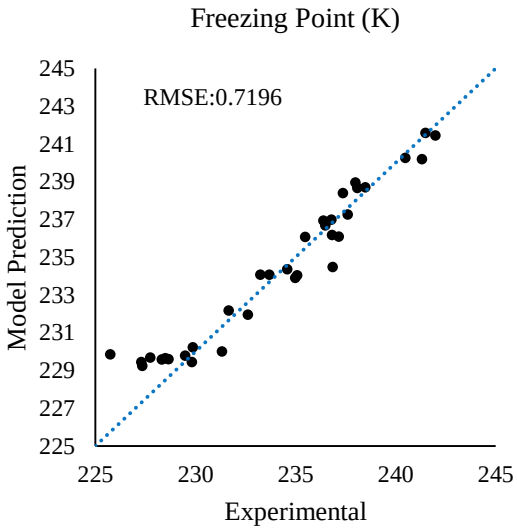

Figure 30 Freezing point neural network comparison of experimental and model predictions.

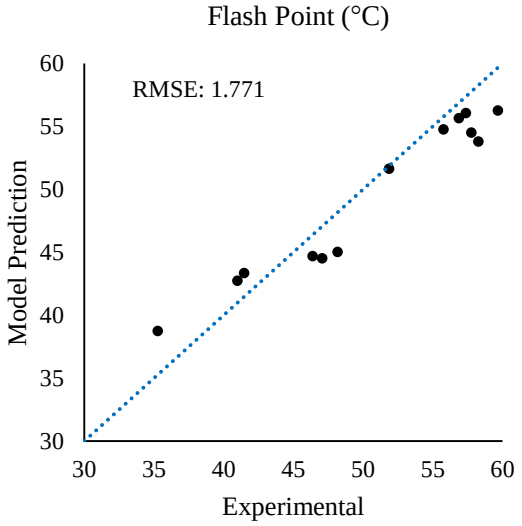

Figure 31 Flash point neural network comparison of experimental and model predictions.

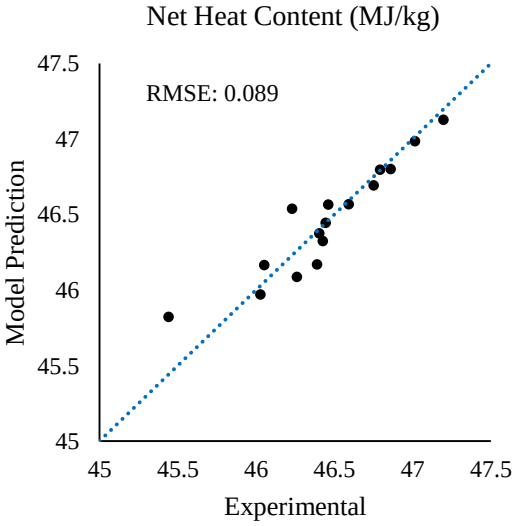

Figure 32 Heat content neural network comparison of experimental and model predictions.

CHAPTER V
REGRESSION ANALYSIS: KRIGING INTERPOLATION

V.1 Kriging Interpolation: Introduction

For the geosciences and created in 1951, Kriging has become a spatial interpolation method to solve optimization problems in any field of study. (Carugo and Eisenhaber 2010) As with any interpolation method, Kriging modeling constructs a model that passes through the training data it is given. This does not imply, however, that the algorithm can only produce one result. Kriging creates stochastic outputs, which take on many different functional forms. (Ginsbourger, et al. 2015) Monte Carlo methods are required to derive approximate solutions. Results from Kriging, while robust, have long simulation times.

V.2 Kriging Interpolation: Parameter Analysis

Each simulation model will always pass through the training data; predictions between the training points are ambiguous and are the unknown elements in the algorithm. (Ginsbourger, et al. 2015) For this reason, it is especially important to optimize this process for effective results. Optimization of this nature does not have a systematic, or solvable, algorithm; the complexity does not allow it. (Hasan 2015) Monte Carlo simulations, however, are a reliable method of increasing performance– albeit a time consuming process. In R, the package DiceOptim was used to generate Kriging models. (Ginsbourger, et al. 2015) The covariance function Matern 5/2 was used. Density, with its linear character, has trivial error associated with it, as shown in Figure 33. For density, 32 sample points were used as training data.

Freezing point has a noticeable difference, as visualized in Figure 34; it benefits from the amount of training data the algorithm is given. For freezing point 32 sets of data points were used as training data.

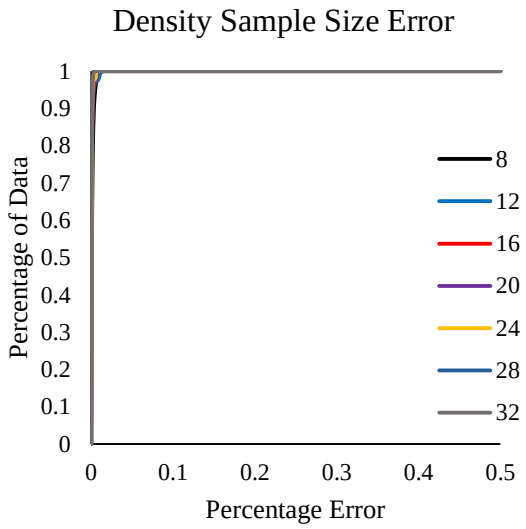

Figure 33 Error plot of density for various sample sizes.

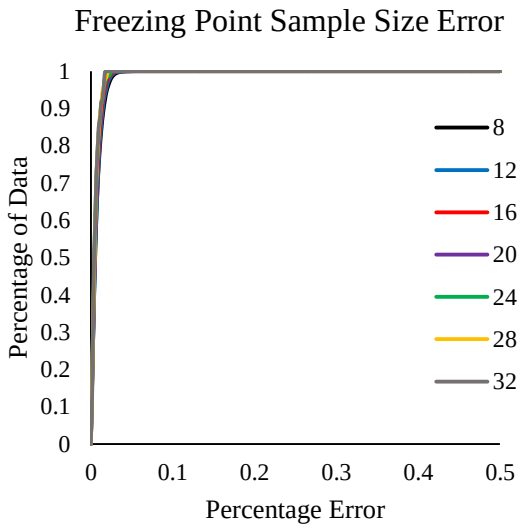

Figure 34 Error plot of freezing point for various sample sizes.

Heat content training size is especially important; within the context of this project, it highly sensitive, due to the low number of data points, shown in Figure 35. For heat content, 12 data points were used for modeling.

Flash point, for Kriging modeling, has a counterintuitive result; 8 samples are better than 12, as shown in figure 36. There are multiple explanations for this. Firstly, and most importantly, the amount of training data is low. Another factor is the fact that flash point is a nonlinear model. Lastly, the concept of white noise exists in any data set. These issues cause; poor confidence in model; makes the model intrinsically difficult to model; and white noise and true data are not discernible due to the low data set provided. This is especially true for an interpolation algorithm. For these reasons, the utilization of too many points is not an optimal model. Flash point modeling was performed with 8 sample points.

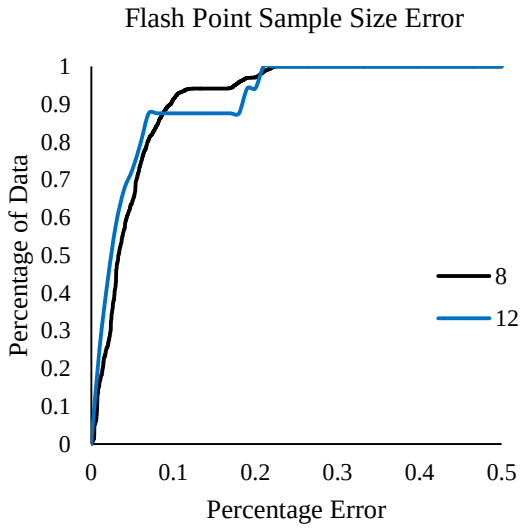

Figure 35 Error plot of flash point for various sample sizes.

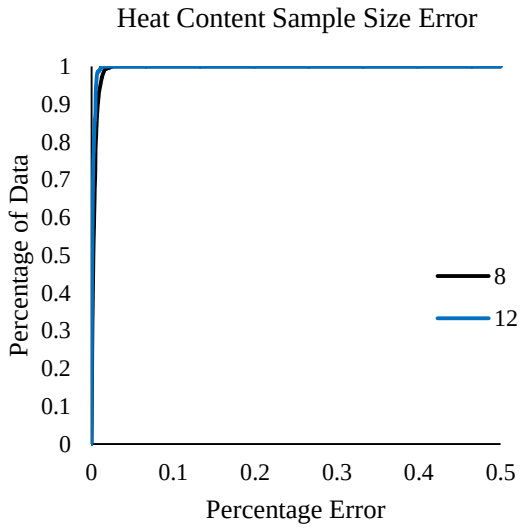

Figure 36 Error plot of heat content for various sample sizes.

V.3 Kriging Interpolation: Monte Carlo Simulations

The statistical programming language, R, was used for Kriging optimization. Simulations that were run with default simulations did not provide adequate results. The parameter θ is stochastically generated for each simulation and has a large impact on the results. Many of which generate results that are too poor for adequate predictions. Elucidating the best parameters for each physical property was a systematic process. A Monte Carlo optimization scheme was coded and simulated using the R package DiceOptim, shown in Figure 37. (Ginsbourger, et al. 2015)

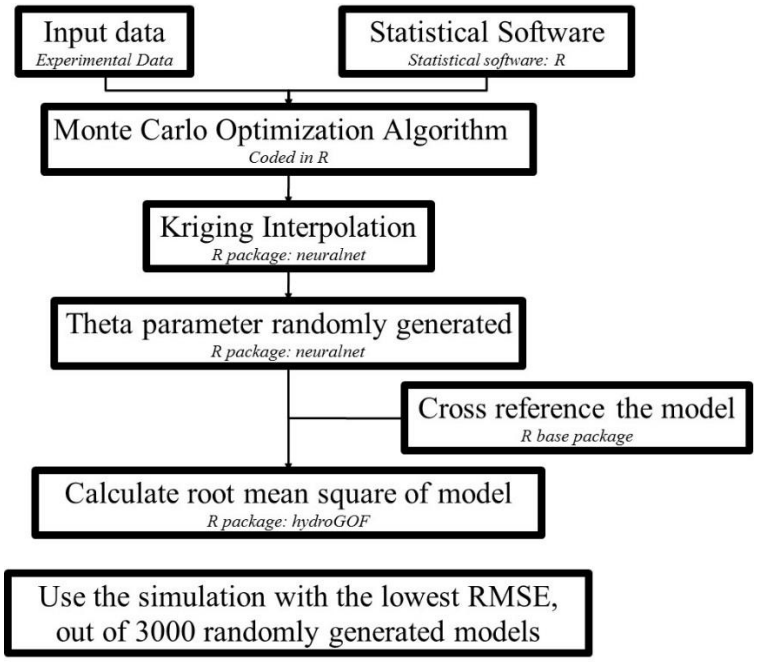

Figure 37 General Optimization scheme for Kriging interpolation.

This process was repeated for 1000 sets of unique θ values. With 1000 θ sets, the root mean square was calculated for each simulation set. The θ set with the lowest root mean square, for the respective physical properties were used and compared. Error plots were constructed for this. The plots compared the non-optimized Kriging simulations against the optimized θ parameters. Results, especially the high error models, increased confidence by a substantial amount.

With the optimal parameters, one more set of Kriging simulations were performed– this was done to further refine the model. While changes in error appear to be trivial, any decrease in root mean square error is highly beneficial to the efficacy of Kriging interpolation.

Density and freezing point have lower errors from optimization, shown in Figure 38 and Figure 39. But density, from a conceptual standpoint, does not gain any efficacy from a pragmatic standpoint; its physical properties exhibit a simplistic linear model that will always create accurate predictions. Conversely, freezing point gains in accuracy is an important parameter to optimize.

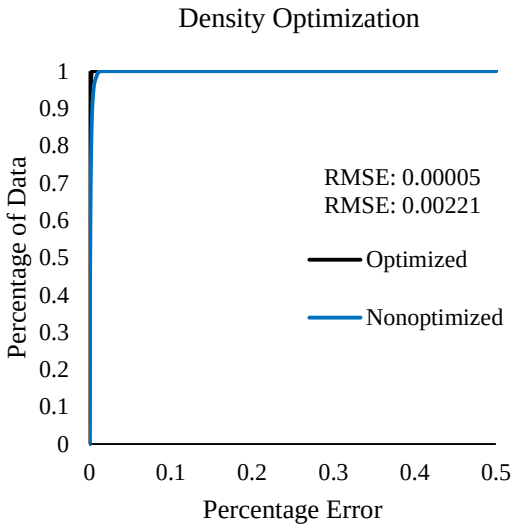

Figure 38 Graphical visualization of density optimization efficacy.

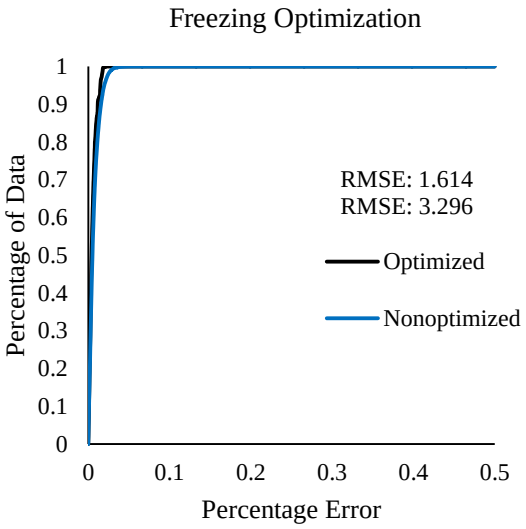

Figure 39 Graphical visualization of freezing point optimization efficacy.

Flash point and heat content have substantial increases in performance from optimization. These results are shown in Figure 38 and Figure 39, respectively. This is an expected result; an interpolation algorithm relies heavily on training data. A low number of runs exacerbate this issue.

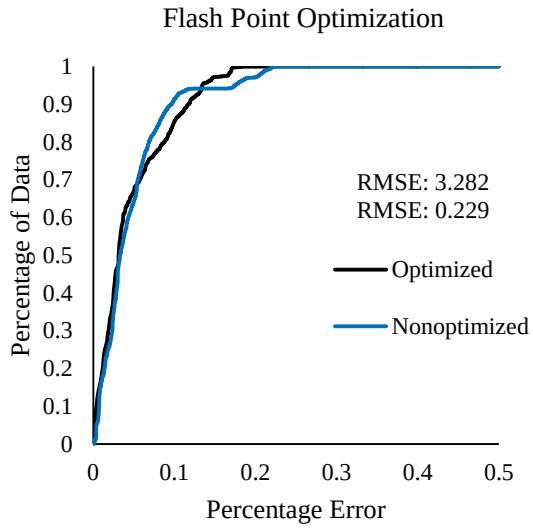

Figure 40 Graphical visualization of density optimization efficacy.

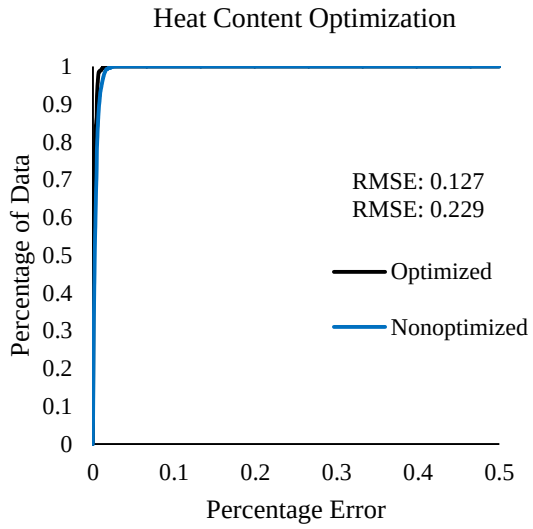

Figure 41 Graphical visualization of freezing point optimization efficacy.

V.4 Kriging Interpolation: Results

With the parameters optimized and assessed, experimental values were compared to the predicted values. The optimization package DiceOptim, generated the following predictions. The work by Al-Nuaimi et al. 2014 is fundamentally different from Kriging interpolation. (Al-Nuaimi, et al. 2014) The careful analysis of training size to use is shown in Table 5. The theta parameters, the most important set of constants, are integrated in the code in R, the programming language.

Table 5 Sample sizes used for each respective physical property for simulations.

	Density	Freezing Point	Flash Point	Heat Content
Training Size	32	32	8	12

Density is accurately quantified; this is the important aspect of this research. The other physical properties, exhibit highly nonlinear models and are more difficult to model; freezing point, heat content, and flash point. The following scatterplots cumulatively represent the results of the implementation of this interpolation method, Figure 42, Figure 43, Figure 44 and Figure 45, respectively.

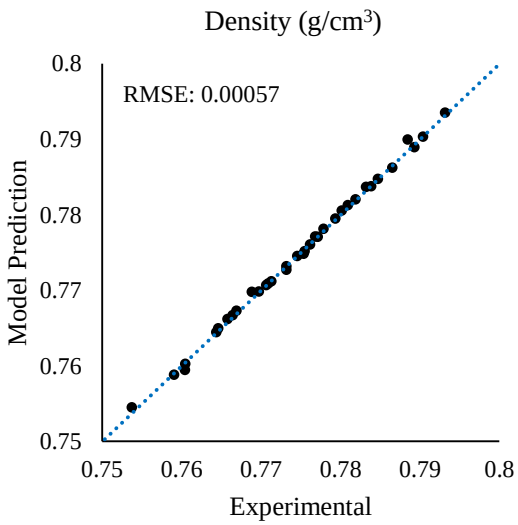

Figure 42 Density scatterplot, comparing the model predictions and experimental values.

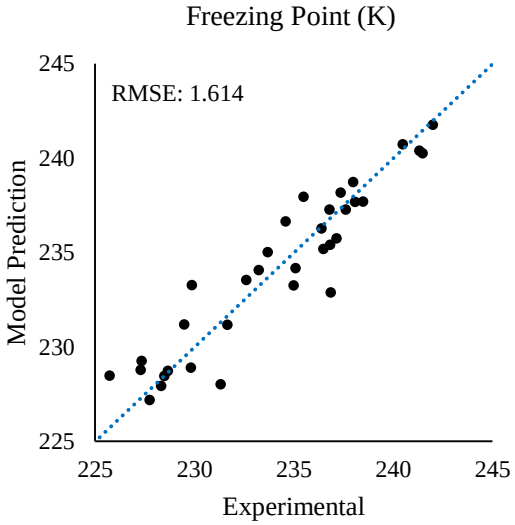

Figure 43 Freezing Point scatterplot, the comparing model predictions and experimental values.

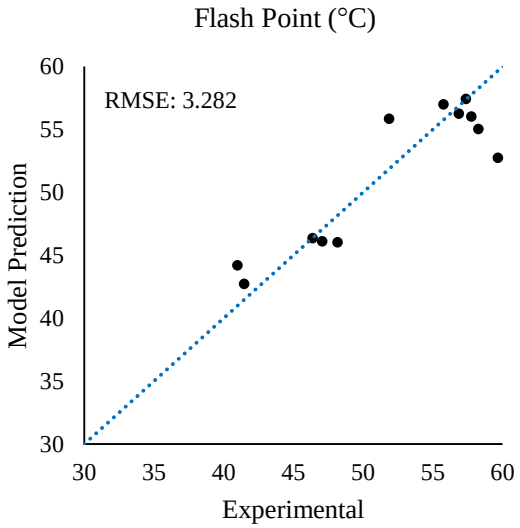

Figure 44 Flash point scatterplot, comparing the model predictions and experimental values.

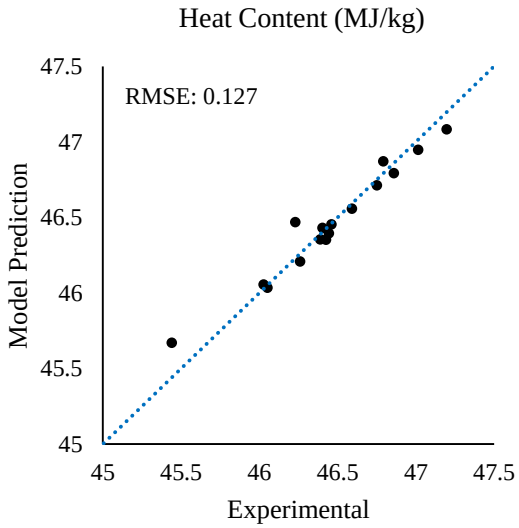

Figure 45 Heat content scatterplot, comparing the model predictions and experimental values.

CHAPTER VI
REGRESSION ANALYSIS: SUPPORT VECTOR MACHINE

VI.1 SVM: Introduction

Support vector machine, or SVM, is another state-of-the-art predictive algorithm, introduced in 1992. (Meyer 2004) Its strength lies in its ability to generated accurate results– even when high dimensional and complex data. SVM is a kernel function, whose algorithmic data analysis relies on dot products. (Carugo and Eisenhaber 2010) One advantage a kernel type function has, is that it distinctly classifies the general trend of the data in question.

Tuning an SVM model for a specific data set relies on the kernel used. The four kernels are linear, polynomial, radial, and sigmoidal. The parameters tested for optimal results were the dimension count, kernel, and SVM algorithm. These parameters are important for reliable and conclusive results. Each individual simulation generates a unique set of supports vectors, or weightage parameters, for each specific physical property in question.

VI.2 SVM: Parameter Analysis

The numbers of dimension tested were 1 through 15. Results fluctuate by substantial amounts, based on the dimension used. This is expected, as the dimension count creates a fundamental difference in how the algorithm processes the data provided. (Meyer 2004)

Two core SVM classifications exist: type 1 (epsilon-SVM), and type 2 (nu-SVM). (Carugo and Eisenhaber 2010) These are minimization techniques, which take on unique functional forms.

Epsilon-SVM Regression

$$\frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i \qquad (6.1)$$

$$where\ y_i(w^T\phi(x_i) + b) \geq 1 - \xi_i\ and\ \xi_i \geq 0, i = 1, \ldots, N \quad (6.2)$$

nu-SVM Regression

$$\frac{1}{2}w^T w - v\rho + \frac{1}{N}\sum_{i=1}^{N}\xi_i \quad (6.3)$$

$$where\ y_i(w^T\phi(x_i) + b) \geq \rho - \xi_i,\ \ \xi_i \geq 0, i = 1, \ldots, N\ and\ \rho \geq 0 \quad (6.4)$$

Where w are the weightages, C is the capacity constant, b a constant, ξ_i the number of inputs, and N the amount of training data used, and ϕ the kernel used.

Optimal parameters were tested systematically, based on algorithm type (epsilon or nu regression), and kernel basis (linear, polynomial, radial, and sigmoidal). The various algorithms are entirely distinct and situational; basic linear functions have the potential to give detrimentally poor results, if a suboptimal kernel is used.

Table 6 Various kernel types in the package e1071.

Kernel	Formula	Parameters
Linear	$u^T v$	none
Polynomial	$\gamma(u^T v + c_0)^d$	γ, d, c_0
Gaussian Radial basis factor	$exp[\lvert-\gamma\rvert u - v^2]$	γ
Sigmoid	$tanh[\gamma\lfloor u^T v + c_0\rfloor]$	γ, c_0

The general optimization scheme is shown in Figure 46. Sample size utilized were the same as the artificial neural network model.

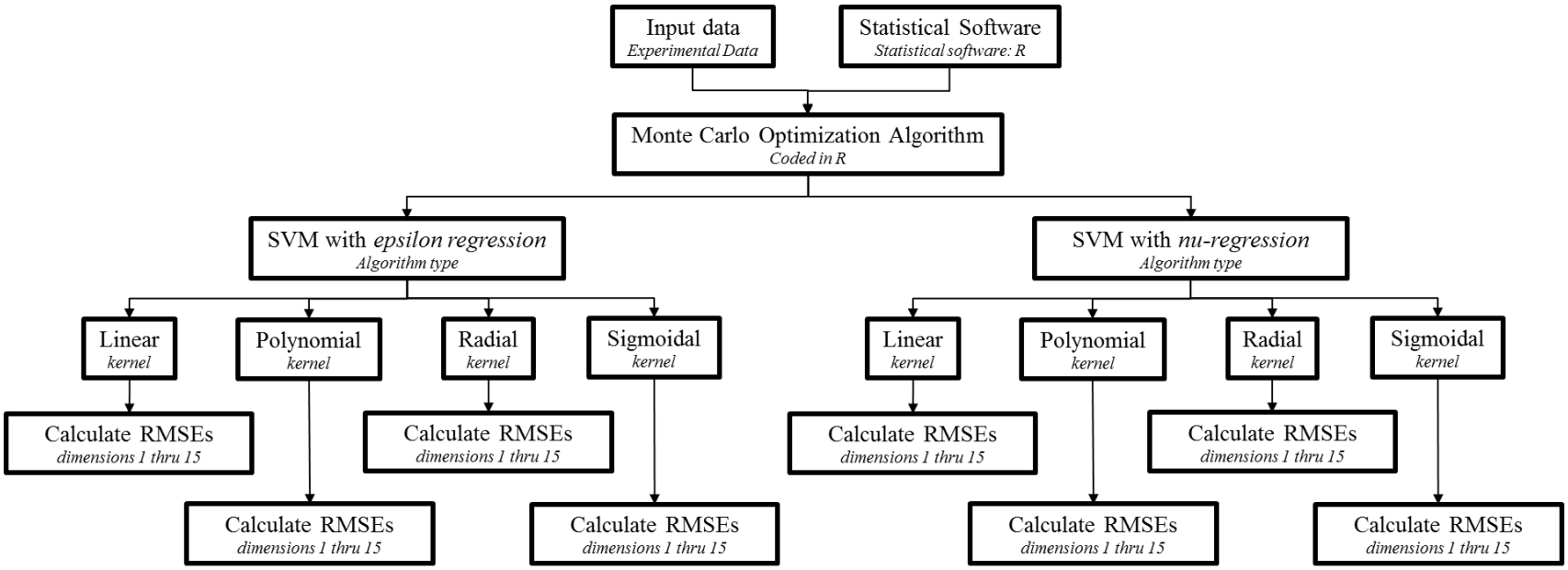

Figure 46 Support vector machine optimization scheme.

Density's optimal algorithm and kernel is linear eps-regression with 3 dimensions. A linear kernel was expected; it was previously concluded that density exhibits a basic trend. The culmination of this analysis is shown in Table 7, Figure 47 and Figure 48.

Table 7 Root mean squares of outputs of each algorithm for density.

Density	Dimensions	eps RMSE	Dimensions	nu RMSE
Linear	3	0.000341375	4	0.000344855
Poly	7	0.006190552	15	0.006406131
Radial	11	0.003791712	2	0.003981989
Sigmoidal	4	0.006315641	6	0.006935549

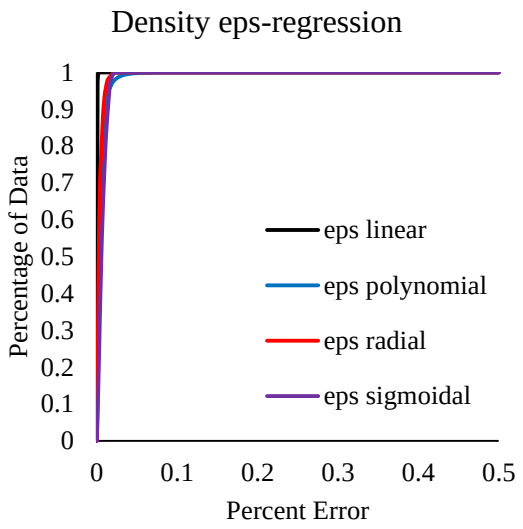

Figure 47 Density error plot of various kernels with their respective optimal dimensions, with eps-regression.

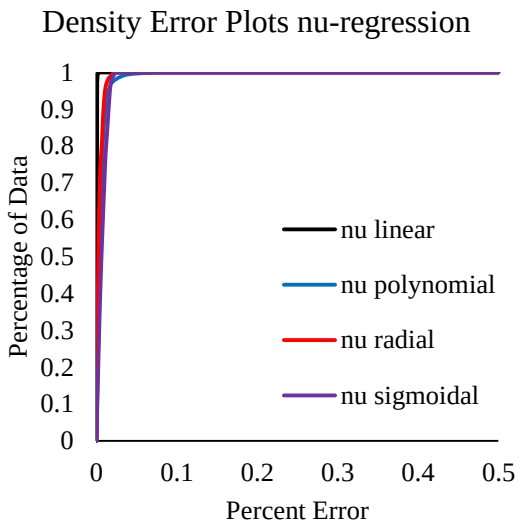

Figure 48 Density error plot of various kernels with their respective optimal dimensions, with nu-regression.

Freezing point, as a whole, is a difficult physical property to model. This physical property exhibits a nonlinear trend. For this reason, optimization is especially important. It was found that a linear eps-kernel with 6 dimensions yielded the best results. The results of these analyses are shown in Table 8, Figure 49 and Figure 50.

Table 8 Root mean squares of outputs of each algorithm for freezing point.

Freezing Point	Dimensions	eps RMSE	Dimensions	nu RMSE
Linear	6	1.275479717	8	1.279689793
Poly	15	3.507836788	1	1.304622813
Radial	4	1.965541944	5	2.013439738
Sigmoidal	8	2.680739371	13	3.045611618

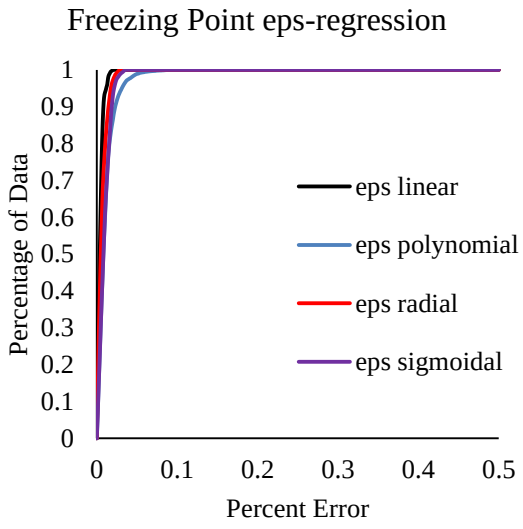

Figure 49 Freezing point error plot of various kernels with their respective optimal dimensions, with eps-regression.

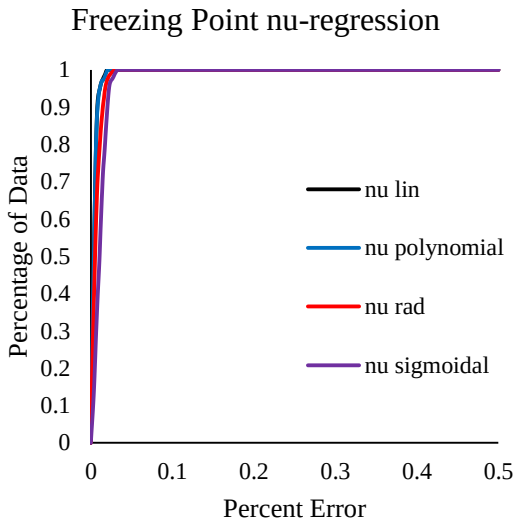

Figure 50 Freezing point error plot of various kernels with their respective optimal dimensions, with nu-regression.

Al Nuaimi et al., 2015 had difficulty in utilizing polynomial regression for flash point predictions. This alludes to flash point exhibiting a highly nonlinear trend. In conjunction with the small amount of training data, this creates models whose predictions are less reliable. SVM, a robust regression method, suffers from lower training data counts. SVM will rarely overfit. Its lack of overfitting, however, will create poor models. (Hoak 2010) The lowest RMSE generated was from radial eps-regression with 14 dimensions. The results of these analyses are shown in Table 9, Figure 51 and Figure 52.

Table 9 Root mean squares of outputs of each algorithm for flash point.

Flash Point	Dimensions	eps RMSE	Dimensions	nu RMSE
Linear	15	4.574605041	5	4.559646099
Poly	14	63.43470901	1	44.93120231
Radial	14	3.86862062	14	4.000084302
Sigmoidal	2	6.879441368	6	5.827137064

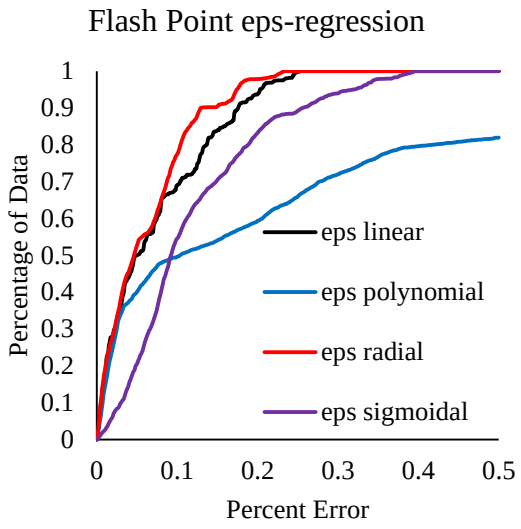

Figure 51 Flash Point error plot of various kernels with their respective optimal dimensions, with eps-regression.

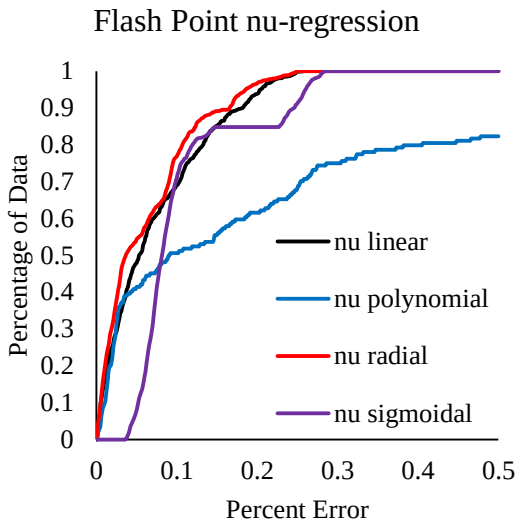

Figure 52 Flash point error plot of various kernels with their respective optimal dimensions, with nu-regression.

Heat content, which also exhibits a nonlinear trends, appears to be well correlated with a linear eps-regression, with 14 dimensions. The results of these analyses are shown in Table 10, Figure 53 and Figure 54.

Table 10 Root mean squares of outputs of each algorithm for heat content.

Heat Content	Dimensions	eps RMSE	Dimensions	nu RMSE
Linear	14	0.188409085	4	0.197230471
Poly	1	5.959983542	5	4.670240029
Radial	8	0.272442733	7	0.278521392
Sigmoidal	2	0.309891269	2	0.306681479

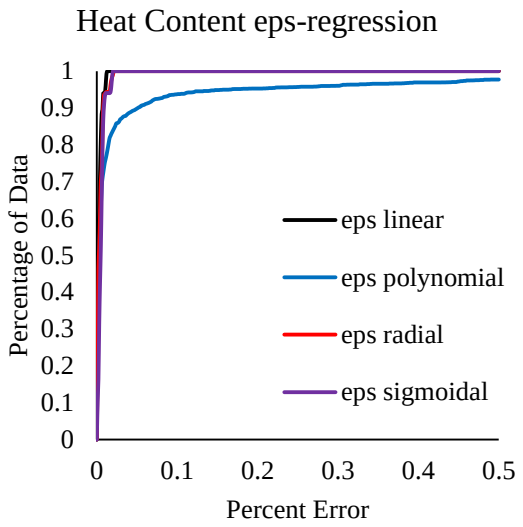

Figure 53 Heat content error plot of various kernels with their respective optimal dimensions, with eps-regression.

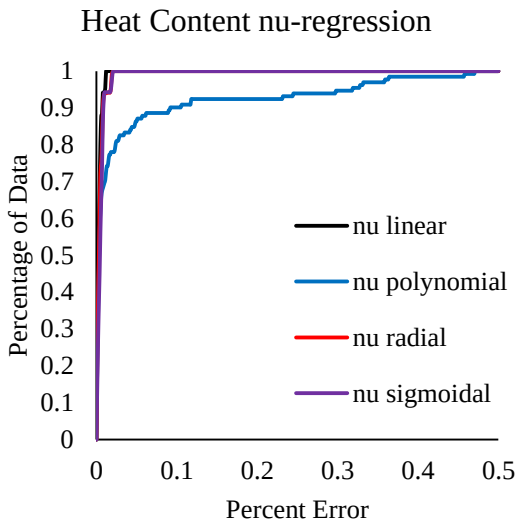

Figure 54 Heat content error plot of various kernels with their respective optimal dimensions, with nu-regression.

VI.3 SVM: Parameter Results

With optimized parameters in e1071, the R package, simulations were performed. (Meyer 2004) As an overview of SVM analysis, the parameters used are shown in Table 11.

Table 11 Established optimal function parameters in e1071.

Physical Property	Kernel	Dimensions	Algorithm	Sample Size
Density	linear	3	eps-regression	32
Freezing Point	linear	6	eps-regression	32
Heat Content	linear	14	eps-regression	12
Flash Point	radial	14	eps-regression	8

With the optimal parameters established, simulations were performed for each physical property; Density, Freezing Point, Heat Content and Flash Point in Figure 55, Figure 56, Figure 57 and Figure 58, respectively.

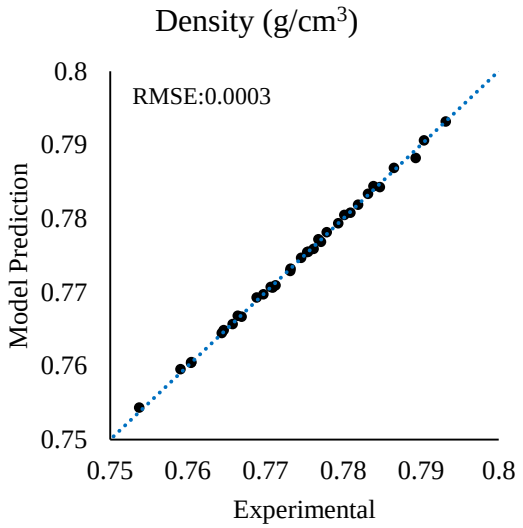

Figure 55 Density scatterplot comparing the model predictions and experimental values.

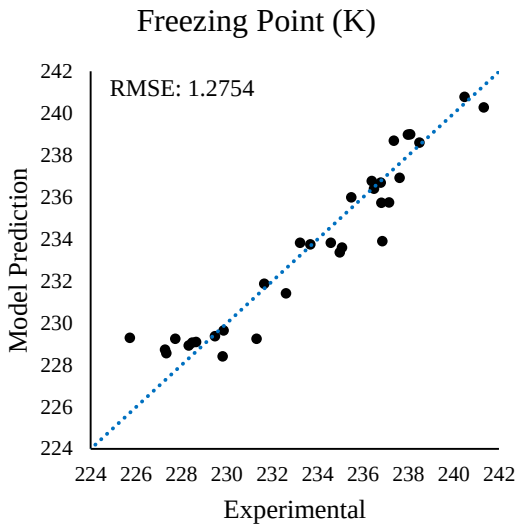

Figure 56 Freezing Point scatterplot comparing the model predictions and experimental values.

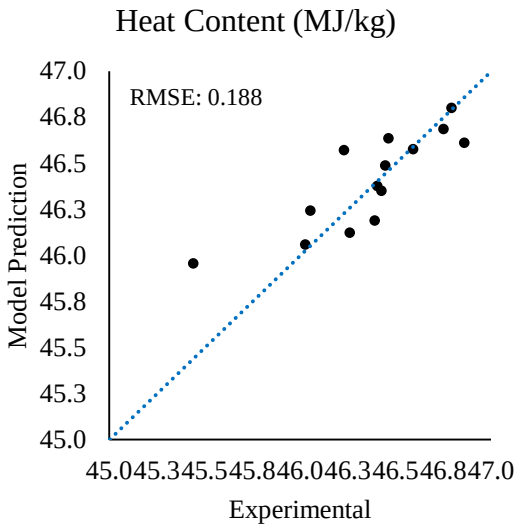

Figure 57 Heat Content scatterplot comparing the model predictions and experimental values.

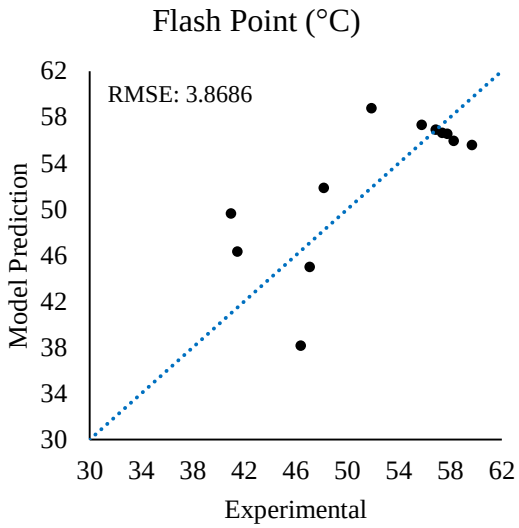

Figure 58 Flash Point scatterplot comparing the model predictions and experimental values.

CHAPTER VII
RESULTS AND DISCUSSION

VII.1 Artificial Neural Network Discussion

Artificial neural networks are a conventional and widely used algorithm for predictive regression modeling. The results vary with each respective physical property, as expected. The number of hidden nodes is considered to be an important parameter during optimization. (Fritsch, Guenther and Suling 2012) But as the number of simulations increases, any specified node converge to equally accurate models. These results allude to a sense of trivialism for this parameter for the function neuralnet in R. (Fritsch, Guenther and Suling 2012) This concept does not apply to all predictive systems or trends.

VII.2 Kriging Interpolation Discussion

Kriging, an interpolation method, is different from a fundamental standpoint– when compared to SVM and neural networks. The number of simulations performed is important to consider for effective utilization. Each simulation created models that had fluctuating ranges in accuracy; it is not as consistent when compared to neural networks and SVM. With increasing simulation count, however, Kriging accuracy produced a model with reliable results.

VII.3 SVM Discussion

SVM, despite being the newest algorithm introduced into the field of statistics, it is not explicitly the best for this research's data. SVM has a predisposition to not overfit. (Hoak 2010) For conventional usage, this is a great strength SVM– data mining methods are utilized with large datasets that tend to generate highly accurate models. Freezing point and density, with their larger datasets, make good candidates for this data mining method. But flash point and heat content do not benefit from this– it may even be argued that it is a detriment. The biggest strength of SVM, as a principle, is its weakness for flash point and heat content modeling. The number of dimensions were not as impactful

as originally expected. This is most likely reflected by the number of simulations performed. As the number of simulations increases, all particular dimensions will converge to it an optimal model. Kernel type, however, was an important decision, which required exhaustive analysis.

VII.4 Calculation Efficiency

From a pragmatic standpoint, calculation and run times were considered, shown in Table 12. Parameters used directly affect this. Simulation count and sample size change the calculation time by a considerable amount. Increasing simulation count increases confidence of results, but at the cost of long simulation time. 1000 simulations were performed for every algorithm and physical property. The CPU used was the Intel(R) Core™ i5-2500 CPU, running at 3.3 GHz. The implementation of each respective regression algorithm yielded similar results.

Table 12 Simulation times for each respective algorithm, with a sample size of 32 and test size of 4.

Algorithm	Single Simulation	Simulations	Total Time
Kriging	.389 mins	1000	~389 mins
ANN	.383 mins	1000	~383 mins
SVM	.395 mins	1000	~395 mins

CHAPTER VIII
CONCLUSION

VIII.1 Overview

This research discusses and reviews the potential to apply advance statistical algorithms to hydrocarbon analysis. Multiple methods were considered; which all produced similar results. The most important part of this project is the creation of a methodology to solve problems in an established field of research. The accuracy and results of each physical property correlates well to the intrinsic strengths and weaknesses of each predictive algorithm.

Conceptual understanding of three distinct modeling techniques have been formulated; along with a functional set of code for actual implementation. All of this work was performed in R, the statistical programming language. A front end package was coded, named DataMine. The package allows complete automation of optimization and implementation.

VIII.2 Recommendations

Future work is a straightforward discussion. More data is required to produce better models. In its current state, and impending future, more experimental work is required. Another point to note is that white noise is pervasive and unavoidable in any regression method. (Hoak 2010) It is important to not, unintentionally, incorporate these deviations into a predictive model. More data directly and easily improves results. A plethora of functions and algorithms exist. Any may be used, but all require more data for accurate physiochemical modeling of fuel systems.

REFERENCES

Al-Nuaimi, Ibrahim A, Rehan Hussain, Mahmoud M. El-Halwagi, and Nimir O. Elbashir. *A Property Integration Approach to Optimizing Jet Fuel Blends from Petroleum- and GTL- Based Fuels.* Chemical Engineering Program, Texas A&M University at Qatar, Artie McFerrin Chemical Engineering Department, 2014.

Aslett, Louis. *Data Mining Lab 7: Introduction to Support Vector Machines (SVMS).* Oxford, England: University of Oxford, 2015.

Bohra, Moiz, Muaz Selam, Mohammed Hafis, Wajdi Ahmed, Mansoor Al-Shamri, and Nimir O. Elbashir. *Role of Aromatics in Synthetic Jet Fuels.* Chemical Engineering Program, Texas A&M University at Qatar, Artie McFerrin Chemical Engineering Department, 2014.

Carugo, Oliviero, and Frank Eisenhaber. "Data mining techniques for the life sciences." Humana Press, 2010.

Fritsch, Stefan, Frauke Guenther, and Marc Suling. "neuralnet: Training of neural networks. R package version 1.32." *http://CRAN.R-project.org/package=neuralnet.* 2012.

Ginsbourger, D., V. Picheny, O. Roustant, and S. Marmin C. Chevalier. "DiceOptim: Kriging-Based Optimization for Computer Experiments. R package version 1.5." *http://CRAN.R-project.org/package=DiceOptim.* 2015.

Hasan, M. M. Faruque, interview by Andrew Yueh. *Kriging Interpolation Discussion* (June 2015).

Hoak, Josh. "The Effects of Outliers on Support Vector Machines." *Portland State University*, April 13, 2010.

Meyer, David. "Support vector machines: The interface to libsvm in package e1071." *http://CRAN.R-project.org/package=e1071.* 2004.

Opec. *Proven Gas Reserves.* December 2013.

Pearce, Brian. "The Shape of Air Travel Markets over the Next 20 Years." *International Air Transport Association*, Februrary 18, 2015.

APPENDIX A

DATAMINE DOCUMENTATION

A.1 errorplot

Description
A plotting function. Generates an error table of the simulation function's optimal model.
Usage
errorplot(table, maxerror = 0.5, datapoints = 100, plot = FALSE)
Arguments

table	Define the variable of krigSim, neuralSim, or svmSim
maxerror	The maximum amount of percent error to be plotted
datapoints	Number data points generated, between 0 and the maxerror
plot	A TRUE/FALSE argument for plotting
crossREF	Number of times to cross reference each model

Value
A data.frame containing percent error with respect to bulk of data with the error

A.2 fuelComp

Description
A function that creates a data table with the permutations of possible fuel blends and their net molecular constituents.
Usage
fuelComp(fuels, incriments = 0.05, fuelratio = FALSE)

Arguments

fuels	data.frame where rows are molecular components of each fuel and the columns are the distinct fuels to be blended.
increments	The step size between blend ratios. Default is .05 incrimental changes.
fuelratio	A TRUE/FALSE argument. Whether or not to incorporate the percent blends in the final data table.

Description
fuelComp can handle, at most, 4 distinct fuels, but any number of hydrocarbons(rows). When utilizing the outputs from fuelComp in the prediction process, make sure you set fuelratio to FALSE or you will get an error.
Value
A data.frame containing fundamental molecular distributions of various blends

A.3 krigComp

Description
A function that takes the optimized model and predicts values based on user-specified input values.
Usage
krigcomp(model, data)

Arguments

model	Model from krigSim.
data	Input data.

Description
A function that takes the optimized model and predicts values based on user-specified input values.
Value
Predicted outputs.

A.4 krigOpt

Description
A function that utilizes Kriging interpolation for simulation-based optimization. Training data is incorporated into a Monte Carlo simulation optimization scheme. Which is then used to predict hydrocarbon blends, as defined by the user.
Usage

```
krigOpt(data = c("density.csv"), fuelcomp = x, inputCOLs = 2:5,
  outputCOL = 6, simCOUNT = 10, crossREF = 10, thetaUpper = c(5, 5, 5,
  5), thetaLower = c(0.01, 0.01, 0.01, 0.01), ignorewarns = FALSE)
```

Arguments

data	A list of data files to be read.
fuelcomp	A data.frame of hydrocarbon blends of interest. This data is extracted from the function 'fuelComp'.
inputCOLs	The columns in each .csv file with the inputs of each respective dataset. All datasets used must be formatted in the same fashion.
outputCOL	The columns in each .csv file with the output of each respective dataset. All datasets used must be formatted in the same fashion.
simCOUNT	The number of simulations to perform in the Monte Carlo optimization scheme.
crossREF	The number of cross referencing to be done in each respective simulation.
thetaUpper	The maximum value of the theta parameter that may be used.

thetaLower The minimum value of the theta parameter that may be used.

ignorewarns Ignore initial checking of user inputs for potential fatal errors during simulation.

samplesize The amount of data used for training with the other points used for testing. Can be a list for each data file or a percentage of the total data in each file. Default is .85 for training.

Details

First, krigOpt handles sample size optimization and then a subsequent Monte Carlo optimization scheme to find the optimal parameters. This is then applied to the user-input data.frame to generate optimal predicted values.

Note

inputCOLs and outputCOL are defined as the column number. They are not defined as a data.frame column. All of the files must be formatted the same, in this regard. The following preliminary error checks are performed:

The data files you put in to the function must exist as a list and as characters. Ex: data=c(\'density.csv\', \'freezing.csv\').

For krigOpt, you are only supposed to define the columns in which all of the values lie. NOT a data.frame like krigSim.

The functions in DataMine cannot handle more than one output at a time! Just define them as separate datasets under data=.

The number of variables in fuelcomp= do not match the number of variables in inputCOL=. You probably left the fuel blend percentages in the data.frame (from the function fuelComp). Take it out of your data.frame.

A.5 krigSim

Description

A Monte Carlo optimization scheme which utilizes DiceOptim to create simulations and find the optimal predictive model.

Usage

```
krigSim(inputCOLs = data[, 2:5], outputCOL = data[, 6], simCOUNT = 1000,
  crossREF = 1000, samplesize = 4, thetaUpper = c(5, 5, 5, 5),
  thetaLower = c(0.01, 0.01, 0.01, 0.01, ignorewarns = FALSE))
```

Arguments

inputCOLs Input values.

outputCOL Objective values.

simCOUNT Number of simulations to perform.

crossREF Number of times to cross reference each model.

samplesize The number of data points used to train the Kriging model. Unused points go to testing.

thetaUpper Upper bound of thetas values allowed.

thetaLower Lower bound of thetas values allowed.

Value
A list containing $predicted $experimental $RMSE $thetas
Note

Two global variables are generated: Kinputs and Koutputs.

krigSim has preliminary error checks, which are as follows:
You set your samplesize= higher than the amount of data you have.
Your inputs and outputs must be in the form of data.frame().
The functions in DataMine can not handle more than one output at a time.

A.6 neuralcomp

Description
A function that takes your optimized model and predicts values based on user-specified input values.
Usage
neuralcomp(model, data)

Arguments
model Model (from neuralSim)

data Input data

Value
A data.frame of predicted outputs.

A.7 neuralOpt

Description
A Monte Carlo Optimization scheme which utilizes neuralnet to create simulations and find the optimal predictive model.
Usage

```
neuralOpt(data = c("density.csv"), fuelcomp = x, inputCOLs = 2:5,
  outputCOL = 6, simCOUNT = 1000, crossREF = 1000, nodes = 1,
  ignorewarns = FALSE, debug = FALSE, neuralnetFormula = 1)
```

Arguments

data	Character list of data files to be read
fuelcomp	data.frame() of hydrocarbon fuel blends to predict
inputCOLs	Numeric values of which columns contain input data
outputCOL	Numeric values of which column contains the output data
simCOUNT	Number of simulations to perform
crossREF	Number of times to cross reference each model
nodes	A numerical list of nodes to be used for each respective data file
ignorewarns	TRUE/FALSE argument. Turn on/off initial error checks before functions are run.
debug	TRUE/FALSE argument. Runs a basic cycle of user-input parameters with text guidance. Used to check for errors.
neuralnetFormula	An argument to specify a specific formula to solve for neuralnet. Default value is 1, indicating the normal formula

Value
A data.frame containing molecular distributions.

Note
InputCOLs and outputCOL are defined as the column number. They are not defined as a data.frame column. All of the files must be formatted the same, in this regard. The following preliminary error checks are performed:

The data files you put in to the function must exist as a list and as characters. Ex: data=c(\'density.csv\', \'freezing.csv\').
For neuralOpt, you are only supposed to define the columns in which all of the values lie. NOT a data.frame like neuralSim.
The functions in DataMine cannot handle more than one output at a time! Just define them as separate datasets under data=
The number of variables in fuelcomp= do not match the number of variables in inputCOL=. You probably left the fuel blend percentages in the data.frame (from the function fuelComp). Take it out of your data.frame.

A.8 neuralSim

Description
A Monte Carlo Optimization scheme which utilizes neuralnet to create simulations and find the optimal predictive model.

Usage
neuralSim(inputCOLs = 2:5, outputCOL = 6, simCOUNT = 1000,

crossREF = 1000, samplesize = 32, nodes = 7, ignorewarns = FALSE)

Arguments

inputCOLs	Input values
outputCOL	Objective values
simCOUNT	Number of simulations to perform
crossREF	Number of times to cross reference each model
samplesize	The number of data points used to train the Kriging model. Unused points go to testing
nodes	Number of hidden nodes in the model

Value
A list containing $predicted $experimental $RMSE $nodes

Note
Two global variables are generated: NNinputs and NNoutputs.
neuralSim has preliminary error checks, which are as follows:
You set your samplesize= higher than the amount of data you have.
Your inputs and outputs must be in the form of data.frame().
The functions in DataMine cannot handle more than one output at a time.

A.9 scatterplot

Description
A plotting function. Generates a scatterplot table of the average predicted vs. experimental values from a simulation function's optimal model.

Usage
scatterplot(table, plot = FALSE)

Arguments

table	Define the variable of krigSim, neuralSim, or svmSim
plot	A TRUE/FALSE argument for plotting.

Details
scatterplot is an all-in-one function that handles krigSim, neuralSim, and svmSim outputs. For error analysis/visualization.
Value
A data.frame containing experimental vs. predicted values (averaged out).

A.10 svmcomp

Description
A function that takes your optimized model and predicts values based on user-specified input values

Usage
```
svmcomp(model, data)
```

Arguments

model	Model from svmSim
data	Input data

Value
A data.frame of predicted outputs.

A.11 svmOpt

Description
A function that utilizes SVM for simulation-based optimization. Training data is incorporated into a Monte Carlo simulation optimization scheme. Which is then used to predict hydrocarbon blends, as defined by the user.

Usage
```
svmOpt(data = c("FlashPoint.csv"), fuelcomp = fuelcomp, inputCOLs = 2:5,
  outputCOL = 6, kernels = c("radial", "polynomial", "linear", "sigmoid"),
  simCOUNT = 1000, crossREF = 1000, dimensionSim = 1:2,
  ignorewarns = FALSE, debug = TRUE)
```

Arguments

data	A list of data files to be read.
fuelcomp	A data.frame of hydrocarbon blends of interest. This data is extracted from the function 'fuelComp'.
inputCOLs	The columns in each .csv file with the inputs of each respective dataset.
outputCOL	The columns in each .csv file with the output of each respective dataset.
simCOUNT	The number of simulations to perform in the Monte Carlo optimization scheme.
crossREF	The number of cross referencing to be done in each respective simulation.
ignorewarns	Ignore initial checking of user inputs for potential fatal errors during simulation.
debug	Performs a quick cycle through the entire simulation scheme, to test for

	errors prior to full simulations.
samplesize	The amount of data used for training with the other points used for testing. Can be a list for each data file or a percentage of the total data in each file. Default is .85 for training.
thetaUpper	The maximum value of the theta parameter that may be used.
thetaLower	The minimum value of the theta parameter that may be used.

Details
First, svmOpt handles sample size optimization and then a subsequent Monte Carlo optimization scheme to find the optimal parameters. This is then applied to the user-input data.frame to generate optimal predicted values

Value
A data.frame containing the predicted values, set by the user in the argument fuelcomp
Note
inputCOLs and outputCOL are defined as the column number. They are not defined as a data.frame column. All of the files must be formatted the same, in this regard. The following preliminary error checks are performed:

The data files you put in to the function must exist as a list and as characters. Ex: data=c(\'density.csv\', \'freezing.csv\').
For svmOpt, you are only supposed to define the columns in which all of the values lie. NOT a data.frame like svmSim.
The functions in DataMine cannot handle more than one output at a time! Just define them as separate datasets under data=
The number of variables in fuelcomp= do not match the number of variables in inputCOL=. You probably left the fuel blend percentages in the data.frame (from the function fuelComp). Take it out of your data.frame!

A.12 svmSim

Description
A Monte Carlo Optimization scheme which utilizes e1071 to create simulations and find the optimal predictive model.

Usage
```
svmSim(inputCOLs = 2:5, outputCOL = 6, kernel = "polynomial",
  simCOUNT = 1000, dimensions = 3, samplesize = 4, ignorewarns = FALSE)
```

Arguments

inputCOLs	Input values
outputCOL	Objective values
kernel	Kernel type to use
simCOUNT	Number of simulations to perform
dimensions	Number of dimensions to use
samplesize	The number of data points used to train the Kriging model. Unused points go to testing
ignorewarns	Ignore initial checking of user inputs for potential fatal errors during simulation.
crossREF	Number of times to cross reference each model

Details
First, svmOpt handles sample size optimization and then a subsequent Monte Carlo optimization scheme to find the optimal parameters. This is then applied to the user-input data.frame to generate optimal predicted values

Value
A list containing $predicted $experimental $RMSE $weights
A list containing $predicted $experimental $rootmean $SV $kernel $dimensions)

Note
Two global variables are generated: SVinputs and SVoutputs.
svmSim has preliminary error checks, which are as follows:
You set your samplesize= higher than the amount of data you have.
Your inputs and outputs must be in the form of data.frame().
The functions in DataMine can not handle more than one output at a time.

APPENDIX B

DATAMINE CODE

B.1 errotplot Code

```
errorplot <- function(table, maxerror=0.5, datapoints=100, plot=FALSE){
  data <- table
  file <- data.frame(data$experimental, data$predicted)
  Exp <- file[,1]
  Pred <- file[,2]
  Error <- abs(data.frame((Exp-Pred)/Exp))
  rowcount <- nrow(Error)
  intervals <- seq(0, maxerror, 1/(datapoints*2))
  id <- 1:length(intervals)
    errorpoints <- NULL
    for (i in id){
    intervalpoint <- intervals[i]
    errorcount <- Error[Error<intervalpoint,]
    points <- length(errorcount)
    standardized <- points/rowcount
    errorpoints <- rbind(errorpoints, standardized)
  }
  x <- intervals
  y <- errorpoints
  if(plot==TRUE){
  plot(x, y)
  }
  Data <- suppressWarnings(data.frame(intervals, errorpoints))
  names(Data) <- c('experimental', 'predicted')
  Data1 <- Data
  }
```

B.2 fuelComp Code

```
fuelComp <- function(fuels ,increments=.05, fuelratio=FALSE){
  if (is.data.frame(fuels)==FALSE){
  print('User input must be in the form of a data.frame')
  stop('Incorrect data type. Not as.data.frame', call=FALSE)
  }

  if (ncol(fuels) > 4){
  print('fuelComp can only handle up to 4 distinct fuel blends!')
```

```
  stop('Too many columns/fuels in data.frame', call.=FALSE)
  }
fuels <- fuels
fuel_count <- ncol(fuels)
fullgrid <- data.frame()
inc <- increments*100
if (fuel_count==4){
fullgrid <- data.frame()
id <- 0:100
for (i in id){
grid <- expand.grid(a=0:100, b=0:100, c=0:100, d=i)
grid100 <- grid[rowSums(grid)==100,]
grid_inc <- grid100[which(grid100[,1] %% inc==0 & grid100[,2] %% inc==0 &
grid100[,3] %% inc==0 & grid100[,4] %% inc==0),]
names(grid_inc) <- names(fuels)
grid100_percent <- grid_inc/100
fullgrid <- rbind(fullgrid, grid100_percent)
rownames(fullgrid) <- c(1:nrow(fullgrid))
}
}
if (fuel_count==3){
grid <- expand.grid(a=0:100, b=0:100, c=0:100)
grid <- data.frame(grid)
grid100 <- grid[rowSums(grid)==100,]
grid_inc <- grid100[which(grid100[,1] %% inc==0 & grid100[,2] %% inc==0 &
grid100[,3] %% inc==0),]
names(grid_inc) <- names(fuels)
fullgrid <- grid_inc/100
rownames(fullgrid) <- c(1:nrow(fullgrid))
}
if(fuel_count==2){
grid <- expand.grid(a=0:100, b=0:100)
grid <- data.frame(grid)
grid100 <- grid[rowSums(grid)==100,]
grid_inc <- grid100[which(grid100[,1] %% inc==0 & grid100[,2] %% inc==0),]
names(grid_inc) <- names(fuels)
fullgrid <- grid_inc/100
rownames(fullgrid) <- c(1:nrow(fullgrid))
}
  bind <- NULL
  id <- 1:nrow(fullgrid)
  for (i in id){
  a <- fuels[,1]*fullgrid[i,1]
  b <- fuels[,2]*fullgrid[i,2]
```

```
  x <- data.frame(a,b)
  if(fuel_count==3){
  c <- fuels[,2]*fullgrid[i,3]
  x <- data.frame(x, c)
  }
  if(fuel_count==4){
  c <- fuels[,2]*fullgrid[i,3]
  d <- fuels[,2]*fullgrid[i,4]
  x <- data.frame(x, c, d)
  }
  y <- t(rowSums(x))
  bind <- rbind(bind, y)
  }
blend <- fullgrid
blends <- data.frame(bind)
names(blends) <- rownames(fuels)
output <- blends
assign("fuel", new.env(hash = TRUE), envir = .GlobalEnv)
assign("fuelratio", fullgrid, envir = fuel)
if(fuelratio==TRUE){
output <- cbind(fullgrid, output)
}
data.frame(output)
}
```

B.3 krigcomp Code

```
krigcomp <- function(model, data){
  x <- model
  y <- km(formula=~1, design=Kinputs, response=Koutputs,
  optim.method='BFGS', upper=x$thetas, parinit=c(.05, .05, .05, .05),
  lower=x$thetas)
  predictions <- predict(object=y, newdata=data, type='SK')
  z <- data.frame(predictions$mean)
  names(z) <- 'predictions'
}
```

B.4 krigOpt Code

```
krigOpt <- function(data= c('density.csv'), fuelcomp=x, inputCOLs=2:5, outputCOL=6,
simCOUNT=10, crossREF=10, thetaUpper= c(5, 5, 5, 5), thetaLower= c(0.01, 0.01, 0.01,
0.01), ignorewarns=FALSE){
  if (ignorewarns==FALSE){
  if (is.character(data)==FALSE){
```

```
  print('The data files you put in to the function must exist as a list and as characters. Ex:
data=c(\'density.csv\', \'freezing.csv\').')
  stop('data= is not as.character', call.=FALSE)
  }
  if (is.data.frame(inputCOLs==TRUE)){
  print('For krigOpt, you are only supposed to define the columns in which all of the
values lie. NOT a data.frame like neuralSim.')
  stop('inputCOLs is a data.frame', call.=FALSE)
  }
  if (length(outputCOL) != 1){
  print('The functions in DataMine can not handle more than one output at a time! Just
define them as separate datasets under data=')
  stop('outputCOL is more than 1 column', call.=FALSE)
  }
  if (ncol(fuelcomp) != length(inputCOLs)){
  print('The number of variables in fuelcomp= do not match the number of variables in
inputCOL=. You probably left the fuel blend percentages in the data.frame (from the
function fuelComp). Take it out of your data.frame!')
  stop('fuelcomp= is not the same number of inputs as inputCOLs=', call.=FALSE)
  }
  }
  result <- matrix(data=NA, ncol=datafiles, nrow=nrow(fuelcomp))
  datafiles <- length(data)
  id <- 1:datafiles
    for (i in id){
    options(warn=-1)
    Data1 <- read.csv(data[i])
    inputs <- Data1[,inputCOLs]
    outputs <- data.frame(Data1[,outputCOL])
    rows <- nrow(inputs)
    sample_table <- NULL
      yd <- 1:length(incriments)
      for (y in yd){
      incriments <- seq(8, (rows-4), 4)
      ud <- 1:crossREF
        for (u in ud){
        samplesize_0 <- incriments[y]
        testingsize <- rows-samplesize_0
        random_5 <- sample(1:rows, samplesize_0)
        parameters_5 <- inputs[random_5, ]
        objectives_5 <- outputs[random_5, ]
        test_parameters_5 <- inputs[-random_5,]
        test_objectives_5 <- outputs[-random_5,]
```

```
      kriging_0 <- km(formula=~1, design=data.frame(parameters_5),
response=data.frame(objectives_5), optim.method='BFGS', upper=c(5, 5, 5, 5),
parinit=c(.05, .05, .05, .05),lower=c(.01, .01, .01, .01))
      predictions <- predict(object=kriging_0, newdata=test_parameters_5, type='SK')
      experimental <- data.frame(test_objectives_5)
      predicted <- data.frame(predictions$mean)
      PvO <- data.frame(experimental, predicted)
      sample_table <- rbind(sample_table, PvO)
      }
     rmse_PvO <- rmse(sample_table[,1], sample_table[,2])
     sample_rmse <- data.frame(samplesize_0, rmse_PvO)
     names(sample_rmse) <- c('samplesize', paste(data[i], 'RMSE'))
     if (y==1){
     best_sample <- sample_rmse
     }
     if (sample_rmse[1,2] < best_sample[1,2]){
     best_sample <- sample_rmse
     }
     }
   opt_samplesize <- best_sample[1,1]
   testingsize <- rows-opt_samplesize
   print(paste(data[i], 'Optimal samplesize found:', opt_samplesize, 'with an RMSE of:',
print(best_sample[1,2])))
   xd <- 1:simCOUNT
     for (x in xd){ #loop
     random <- sample(1:rows, opt_samplesize)
     parameters <- inputs[random,]
     objectives <- outputs[random,]
     kriging <- km(formula= ~1, design=data.frame(parameters),
response=data.frame(objectives), optim.method='BFGS',
parinit=c(.05, .05, .05, .05),lower=thetaLower, upper=thetaUpper)
     unclass <- unclass(kriging)
     cov.attr <- attr(unclass, "covariance")
     unclass_theta <- unclass(cov.attr)
     thetas <- data.frame(attr(unclass_theta, 'range.val'))
     names(thetas) <- 'thetas'
     c1 <- thetas[1,1]
     c2 <- thetas[2,1]
     c3 <- thetas[3,1]
     c4 <- thetas[4,1]
      table <- matrix(data=NA, ncol=2, nrow=(testingsize*crossREF))
      fd <- 1:crossREF
      for(f in fd){
      dataplacement <- c(1:crossREF)-1
```

```
        q <- dataplacement
        random_0 <- sample(1:rows, opt_samplesize)
        parameters_0 <- inputs[random_0,]
        objectives_0 <- outputs[random_0, ]
        test_parameters_0 <- inputs[-random_0, ]
        test_objectives_0 <- outputs[-random_0, ]
        kriging_0 <- km(formula=~1, design=data.frame(parameters_0),
response=data.frame(objectives_0), optim.method='BFGS', upper=c(c1,c2,c3,c4),
parinit=c(.05, .05, .05, .05), lower=c(c1,c2,c3,c4))
        predictions <- predict(object=kriging_0, newdata=test_parameters_0, type='SK')
        experimental <- data.frame(test_objectives_0)
        predicted <- data.frame(predictions$mean)
        collective_weights <- thetas
        PvO <- data.frame(experimental, predicted)
        table[(q*testingsize+1):(f*testingsize),1] <- PvO[,1]
        table[(q*testingsize+1):(f*testingsize),2] <- PvO[,2]
        }
      data1 <- table
      rmse <- data1
      rootmean <- rmse(data1[,1], data1[,2])
      stats <- cbind(collective_weights, 0)
      stats[1,2] <- rootmean
      colnames(stats) <- c('experimental', 'predicted')
      colnames(data1) <- c('experimental', 'predicted')
      overall_file <- rbind(stats, data1)
      if (i==1){
      overall <- overall_file
      }
      if (overall_file[1,2] < overall[1,2]){
      overall <- overall_file
      }
      }
    thetas <- c(overall[1:4,1])
      sd <- 1:50
      mean_pred <- matrix(data=NA, ncol=length(sd), nrow=nrow(fuelcomp))
      for(s in sd){
      random_0 <- sample(1:rows, opt_samplesize)
      parameters_0 <- inputs[random_0,]
      objectives_0 <- outputs[random_0, ]
      test_parameters_0 <- inputs[-random_0, ]
      test_objectives_0 <- outputs[-random_0, ]
      kriging_0 <- km(formula=~1, design=data.frame(parameters_0),
response=data.frame(objectives_0), optim.method='BFGS', upper=c(c1,c2,c3,c4),
parinit=c(.05, .05, .05, .05), lower=c(c1,c2,c3,c4))
```

```
    predictions <- predict(object=kriging_0, newdata=fuelcomp, type='SK')
    predicted <- data.frame(predictions$mean)
    mean_pred[,s] <- predicted[,1]
    }
   x <- data.frame(rowSums(mean_pred))/ncol(mean_pred)
   names(x) <- data[i]
   result[,i] <- x[,1]
   }
 result <- data.frame(result)
 names(result) <- data
 data.frame(result)
}
```

B.5 krigSim Code

```
krigSim <- function(inputCOLs=data[,2:5], outputCOL=data[,6], simCOUNT=1000,
crossREF=1000, samplesize=4, thetaUpper= c(5, 5, 5, 5), thetaLower= c(0.01, 0.01, 0.01,
0.01, ignorewarns=FALSE)){
library(hydroGOF)
library(DiceOptim)
if (ignorewarns==FALSE){
if (samplesize >= nrow(inputCOLs)){
print('You set your samplesize= higher than the amount of data you have.')
stop('samplesize too high', call.=FALSE)
}
if (is.data.frame(inputCOLs==FALSE)){
print('Your inputs and outputs must be in the form of data.frame(')
stop('inputCOLs not a data.frame', call.=FALSE)
}
if (length(outputCOL) != 1){
print('The functions in DataMine can not handle more than one output at a time!')
stop('More than one column in outputCOL', call.=FALSE)
}
}
Kinputs <<- inputCOLs
inputs <- Kinputs
Koutputs <<- data.frame(outputCOL)
outputs <- Koutputs
rows <- nrow(inputs)
 id <- 1:simCOUNT
 for (i in id){
 random <- sample(1:rows, samplesize)
 parameters <- inputs[random,]
 objectives <- outputs[random,]
```

```
  kriging <- km(formula= ~1, design=data.frame(parameters),
response=data.frame(objectives), optim.method='BFGS',
parinit=c(.05, .05, .05, .05),lower=thetaLower, upper=thetaUpper)
  unclass <- unclass(kriging)
  cov.attr <- attr(unclass, "covariance")
  unclass_theta <- unclass(cov.attr)
  thetas <- data.frame(attr(unclass_theta, 'range.val'))
  names(thetas) <- 'thetas'
  c1 <- thetas[1,1]
  c2 <- thetas[2,1]
  c3 <- thetas[3,1]
  c4 <- thetas[4,1]
    table <- NULL
    fd <- 1:crossREF
      for(f in fd){
      random_0 <- sample(1:rows, samplesize)
      parameters_0 <- inputs[random_0,]
      objectives_0 <- outputs[random_0, ]
      test_parameters_0 <- inputs[-random_0, ]
      test_objectives_0 <- outputs[-random_0, ]
      kriging_0 <- km(formula=~1, design=data.frame(parameters_0),
response=data.frame(objectives_0), optim.method='BFGS', upper=c(c1,c2,c3,c4),
parinit=c(.05, .05, .05, .05), lower=c(c1,c2,c3,c4))
      predictions <- predict(object=kriging_0, newdata=test_parameters_0, type='SK')
      experimental <- data.frame(test_objectives_0)
      predicted <- data.frame(predictions$mean)
      collective_weights <- thetas
      PvO <- data.frame(experimental, predicted)
      table <- rbind(table,PvO)
      }
  data <- table
  rmse <- data
  rootmean <- rmse(data[,1], data[,2])
  stats <- cbind(collective_weights, 0)
  stats[1,2] <- rootmean
  colnames(stats) <- c('experimental', 'predicted')
  colnames(data) <- c('experimental', 'predicted')
  overall_file <- rbind(stats, data)
  if (i==1){
  overall <- overall_file
  }
  if (overall_file[1,2] < overall[1,2]){
  overall <- overall_file
  }
```

```
  }
data1 <- overall[-(1:4),]
rootmean <- overall[1,2]
names(rootmean) <- 'RMSE'
thetas <- data.frame(overall[1:4,1])
names(thetas) <- 'thetas'
c(data1, rootmean, thetas)
}
```

B.6 neuralcomp Code

```
neuralcomp <- function(model, data){
library(hydroGOF)
x <- model
parameters <- data.frame(NNinputs)
objectives <- data.frame(NNoutputs)
norm_object <- data.frame(scale(objectives))
weights <- data.frame(x$weights)
Training <- data.frame(parameters, norm_object)
input_colname <- colnames(parameters)
output_colname <- colnames(norm_object)
NNformula <- as.formula(c(paste(output_colname, '~', paste(input_colname, collapse =
"+"))))
ANN_0 <- neuralnet(NNformula, Training, hidden=x$nodes, threshold=c(0.1), rep=1,
algorithm='rprop+', startweights=weights, stepmax= 2e+05)
comp <- compute(ANN_0, data)
std <- sd(data.matrix(objectives))
mn <- mean(data.matrix(objectives))
predicton <- std*comp$net.result+mn
}
```

B.7 neuralOpt

```
neuralOpt <- function(data= c('density.csv'), fuelcomp= x, inputCOLs=2:5,
outputCOL=6, simCOUNT=1000, crossREF=1000, nodes=1, ignorewarns=FALSE,
debug=FALSE, neuralnetFormula=1){
library(hydroGOF)
library(neuralnet)
library(reshape)
if(ignorewarns==FALSE){
if (is.data.frame(inputCOLs==TRUE)){
print('For neuralOpt, you are only supposed to define the columns in which all of the
values lie. NOT a data.frame like neuralSim.')
stop('inputCOLs is a data.frame', call.=FALSE)
```

```
}
if (length(outputCOL) != 1){
print('The functions in DataMine can not handle more more than one output at a time! Just define them as separate datasets under data=')
stop('outputCOL is more than 1 column', call.=FALSE)
}
if (ncol(fuelcomp) != length(inputCOLs)){
print('The number of variables in fuelcomp= do not match the number of variables in inputCOL=. You probably left the fuel blend percentages in the data.frame (from the function fuelComp). Take it out of your data.frame!')
stop('fuelcomp= is not the same number of variables as inputCOLs=', call.=FALSE)
}
if (neuralnetFormula != 1){
print('Be careful if you set your own formula. If you are running multiple files in neuralOpt, the inputCOLs and outputCOL column names, your variables, must the same. Run debug=TRUE first to test.')
}
}
result <- matrix(data=NA, ncol=length(data), nrow=nrow(fuelcomp))
if (debug==TRUE){
print('Debug forces 5 simulations and 5 cross references for each data file.')
print('Starting...')
simCOUNT <- 5
crossREF <- 5
}
datafiles <- length(data)
  id <- 1:datafiles
  for (i in id){
  if (debug==TRUE){
  print(paste('LINE 57 Initializing optimization of', data[i], '...'))
  }
  options(warn=-1)
  Data1 <- read.csv(data[i])
  if (debug==TRUE){
  print(paste('LINE 63 Setting up data.frames for input and outputs for', data[i]))
  }
  inputs <- Data1[,inputCOLs]
  in_colname <- names(inputs)
  outputs <- data.frame(Data1[,outputCOL])
  out_colname <- names(Data1[outputCOL])
  rows <- nrow(inputs)
  formula <- as.formula(c(paste(out_colname, '~', paste(in_colname, collapse = "+"))))
  if (neuralnetFormula!=1){
  formula <- neuralnetFormula
```

```
  }
  incriments <- seq(8, (rows-4), 4)
  sample_table <- NULL
  yd <- 1:length(incriments)
  print('Calculating sample sizes...')
  for (y in yd){
  ud <- 1:crossREF
  for (u in ud){
  dataset <- c(1:crossREF)-1
  q <- dataset[u]
  samplesize_0 <- incriments[y]
  testingsize <- rows-samplesize_0
  random_0 <- sample(1:rows, samplesize_0)
  parameters_0 <- inputs[random_0, ]
  objectives_0 <- outputs[random_0, ]
  test_parameters_0 <- inputs[-random_0,]
  test_objectives_0 <- outputs[-random_0,]
  norm_object_0 <- data.frame(scale(objectives_0))
  names(parameters_0) <- in_colname
  names(norm_object_0) <- out_colname
  Training_0 <- data.frame(parameters_0, norm_object_0)
  if (debug==TRUE){
  print(paste('LINE 96 [ Samplesize', incriments[y],'Cross Reference', u, ']','Running
neuralnet for', data[i], '...', collapse=''))
  }
  ANN_node <- neuralnet(formula, Training_0, hidden=5,
  threshold=c(0.1), rep=1, algorithm='rprop+', stepmax= 2e+05)
  if (debug==TRUE){
  print(paste('LINE 104 [ Samplesize', incriments[y],'Cross Reference', u, ']','Computing
predictions for testing data for', data[i], '...', collapse=''))
  }
  net.results <- compute(ANN_node, test_parameters_0)
  predicted_0 <- sd(objectives_0)*net.results$net.result+mean(objectives_0)
  sample_resulttable <- data.frame(test_objectives_0, predicted_0)
  sample_table <- rbind(sample_table, sample_resulttable)
  }
  if (debug==TRUE){
  print(paste('LINE 115 [ Samplesize', incriments[y],'Simulation Complete ]','Computing
RMSE between experimental and predicted values for', data[i], '...', collapse=''))
  }
  rmse_PvO <- rmse(sample_table[,1], sample_table[,2])
  sample_rmse <- data.frame(samplesize_0, rmse_PvO)
  names(sample_rmse) <- c('samplesize', paste(data[i], 'RMSE'))
  print(sample_rmse)
```

```
  if (debug==TRUE){
  print(paste('LINE 126 [ Samplesize', incriments[y], ']','Comparing error of current
simulation to previously optimal for', data[i], '...', collapse=''))
  }
  if (y==1){
  best_sample <- sample_rmse
  }
  if (debug==TRUE){
  print(paste('LINE 134 [ Samplesize', incriments[y], ']','Comparing error of current node
simulation to previously optimal for', data[i], '...', collapse=''))
  }
  if (sample_rmse[1,2] < best_sample[1,2]){
  best_sample <- sample_rmse
  }
  }
  opt_samplesize <- best_sample[1,1]
  testingsize <- rows-opt_samplesize
  print(paste(data[i], 'Optimal samplesize found:', opt_samplesize, 'with an RMSE of:',
print(best_sample[1,2])))
  print('Starting node calculations...')
    td <- 1:nodes
    for (t in td){
    gd <- 1:crossREF
    if (debug==TRUE){
    print(paste('LINE 112 Simulation start for neural network model of', data[i], '...'))
    gd <- 1:5
    }
      node_table <- matrix(data=NA, ncol=2, nrow=length(gd)*testingsize)
      for (g in gd){
      q <- c(1:crossREF)-1
      random_1 <- sample(1:rows, opt_samplesize)
      parameters_1 <- inputs[random_1, ]
      objectives_1 <- outputs[random_1, ]
      test_parameters_1 <- inputs[-random_1,]
      test_objectives_1 <- outputs[-random_1,]
      norm_object_1 <- data.frame(scale(objectives_1))
      names(parameters_1) <- in_colname
      names(norm_object_1) <- out_colname
      Training_1 <- data.frame(parameters_1, norm_object_1)
      if (debug==TRUE){
      print(paste('LINE 137 [ Node', t,'Cross Reference', g,']','Running neuralnet for',
data[i], '...', collapse=''))
      }
```

```
    ANN_node <- neuralnet(formula, Training_1, hidden=g, threshold=c(0.1), rep=1,
algorithm='rprop+', stepmax= 2e+05)
    if (debug==TRUE){
    print(paste('LINE 145[ Node', t,'Cross Reference', g,']','Computing predictions for
testing data for', data[i], '...', collapse=''))
    }
    net.results <- compute(ANN_node, test_parameters_1)
    predicted_1 <- sd(objectives_1)*net.results$net.result+mean(objectives_1)
    node_resulttable <- data.frame(test_objectives_1, predicted_1)
    if (debug==TRUE){
    print(paste('LINE 152 [ Node', t,'Cross Reference', g,']','Writing results into table for',
data[i], '...', collapse=''))
    }
    node_table[(q*testingsize+1):(g*testingsize), 1] <- node_resulttable[,1]
    node_table[(q*testingsize+1):(g*testingsize), 2] <- node_resulttable[,2]
    }
  if (debug==TRUE){
  print(paste('LINE 159 [ Node', t,'Simulation ]','computing results between
experimental and predicted values for', data[i], '...', collapse=''))
  }
  rmse_PvO <- rmse(node_table[,1], node_table[,2])
  node_rmse <- data.frame(t, rmse_PvO)
  names(node_rmse) <- c('hidden nodes', paste(data[i], 'RMSE'))
  print(node_rmse)
  if (t==1){
  best_nodes <- node_rmse
  }
  if (debug==TRUE){
  print(paste('LINE 176 [ Simulation', t, ']','Comparing error of current node simulation
to previously optimal for', data[i], '...', collapse=''))
  }
  if (node_rmse[1,2] < best_nodes[1,2]){
  best_nodes <- node_rmse
  }
  opt_node <- best_nodes[1,1]
  }
 if (debug==TRUE){
 print(paste('LINE 185 [ Node Simulation Complete]','Optimal node found for', data[i],
'...', collapse=''))
 }
 print(paste(data[i], 'optimal node count found:', opt_node, 'with an RMSE of:',
print(best_nodes[1,2])))
 print('Starting Monte Carlo scheme with optimized parameters...')
```

```
    xd <- 1:simCOUNT
    if (debug==TRUE){
    print(paste('LINE 193 [ Weight Parameter Simulation Start ]','Starting internal
parameter elucidation for', data[i], '...', collapse=''))
    xd <- 1:5
    }
    for (x in xd){
    random <- sample(1:rows, opt_samplesize)
    parameters <- inputs[random, ]
    objectives <- outputs[random, ]
    norm_object <- data.frame(scale(objectives))
    names(parameters) <- in_colname
    names(norm_object) <- out_colname
    Training <- data.frame(parameters, norm_object)
    if (debug==TRUE){
    print(paste('LINE 210 [ Simulation', x, ']','Running neuralnet for weight parameters of',
data[i], '...', collapse=''))
    }
    ANN <- neuralnet(formula, Training, hidden=opt_node,
    threshold=c(0.1), rep=1, algorithm='rprop+', stepmax= 2e+05)

    weight <- unlist(ANN$weight)
    weight.dataframe <- data.frame(weight)
      fd <- 1:crossREF
      #Debugging
      if (debug==TRUE){
      print(paste('LINE 225 [ Simulation', x, ']','nueralnet initial parameter setup complete.
Cross referencing of', data[i], 'simulation starting...', collapse=''))
      fd <- 1:5
      }
      table <- matrix(data=NA, ncol=2, nrow=length(fd)*testingsize)
      dataplacement <- c(1:length(fd))-1
      for (f in fd){
      dataplacement <- c(1:length(fd))-1
      q <- dataplacement[f]
      weight_nrow <- length(weight)
      randomized_weight_range <- seq(-0.2, 0.2, by=2e-04)
      samples <- sample(-randomized_weight_range, weight_nrow)
      weights_0 <- weight + samples
      random_0 <- sample(1:rows, opt_samplesize)
      parameters_0 <- inputs[-random_0, ]
      objectives_0 <- outputs[-random_0, ]
      norm_object_0 <- data.frame(scale(objectives_0))
      names(parameters_0) <- in_colname
```

```
    names(norm_object_0) <- out_colname
    Training_0 <- data.frame(parameters_0, norm_object_0)
    if (debug==TRUE){
    print(paste('LINE 251 [ Simulation', x, 'Cross reference', f,']','nueralnet calculation
starting for', data[i], '...', collapse=''))
    }
    ANN_0 <- neuralnet(formula, Training_0, hidden=opt_node,
    threshold=c(0.1), rep=1, algorithm='rprop+', startweights=weights_0, stepmax=
2e+05)
    net.results <- compute(ANN_0, parameters_0)
    predicted <- sd(objectives_0)*net.results$net.result+mean(objectives_0)
    collective_weights <- unlist(ANN_0$weight)
    PvO <- data.frame(objectives_0, predicted)
    table[(q*testingsize+1):(f*testingsize), 1] <- PvO[,1]
    table[(q*testingsize+1):(f*testingsize), 2] <- PvO[,2]
    }
  if (debug==TRUE){
  print(paste('LINE 265 [ Simulation', x, ']','Cross referencing complete for', data[i],
collapse=''))
  }
  data_0 <- table
  colnames(data_0) <- c('experimental', 'predicted')
  if (debug==TRUE){
  print(paste('LINE 273 [ Simulation', x, ']','Calculating root mean square error of',
data[i], collapse=''))
  }
  rmse <- data_0
  rootmean <- rmse(data_0[,1], data_0[,2])
  stats <- cbind(weight.dataframe, 0)
  stats[1,2] <- rootmean
  names(stats) <- c('experimental', 'predicted')
  weightlength <- nrow(stats)
  overall_file <- suppressWarnings(rbind(stats, data_0))
  if (debug==TRUE){
  print(paste('LINE 284 [ Simulation', x, ']','Writing data table for', data[i], '...',
collapse=''))
  }
  if (debug==TRUE){print(paste('LINE 286 [ Simulation', x, ']','Comparing error of
current simulation to previously optimal for', data[i], '...', collapse=''))}
  }
 if (i==1){
 overall <- overall_file
 }
 if (overall_file[1,2] < overall[1,2]){
```

```
  overall <- overall_file
  }
  weights <- data.frame(overall[1:weightlength,1])
  if (debug==TRUE){
  print(paste('LINE 301 [ Simulations Complete ]','Optimal model found for', data[i], '...',
collapse=''))
  }
  print(paste('Predicting user-supplied input data for', data[i]))
    sd <- 1:50
    if (debug==TRUE){
    print(paste('LINE 307 [ Simulations Complete ]','Optimal modeling starting for the
fuelinput...', collapse=''))
    sd <- 1:5
    }
    mean_pred <- matrix(data=NA, ncol=length(sd), nrow=(nrow(fuelcomp)))
    for (s in sd){
    random_2 <- sample(1:rows, opt_samplesize)
    parameters_2 <- inputs[random_2, ]
    objectives_2 <- outputs[random_2, ]
    norm_object_2 <- data.frame(scale(objectives_2))
    names(parameters_2) <- in_colname
    names(norm_object_2) <- out_colname
    Training_2 <- data.frame(parameters_2, norm_object_2)
    if (debug==TRUE){
    print(paste('LINE 325 [ Optimal Model Simuation Cross Reference', s, ']','About to
run neuralnet model...', collapse=''))
    }
    ANN_2 <- neuralnet(formula, Training_2, hidden=opt_node, threshold=c(0.1), rep=1,
algorithm='rprop+', startweights=weights, stepmax= 2e+05)
    if (debug==TRUE){
    print(paste('LINE 337 [ Optimal Model Simuation Cross Reference', s, ']','Predicting
values for user input data.frame...'))
    }
    net.results <- compute(ANN_2, fuelcomp)
    predicted <- sd(objectives_1)*net.results$net.result+mean(objectives_1)
    mean_pred[,s] <- predicted[,1]
    }
  if (debug==TRUE){
  print(paste('LINE 347 [', data[i], 'modeling complete', ']', 'Binding and averaging (from
cross referencing) predicted values for user input data.frame...', collapse=''))
  }
  k <- data.frame(rowSums(mean_pred))/ncol(mean_pred)
  result[,i] <- k[,1]
  if(i!=length(data) & length(data)!=1){
```

```
  print(paste(data[i], 'complete. Cycling to', data[i+1], '...'))
  }
}
print(paste('neuralOpt complete'))
result <- data.frame(result)
colnames(result) <- data
data.frame(result)
}
```

B.8 neuralSim Code

```
neuralcomp <- function(model, data){
library(hydroGOF)
x <- model
parameters <- data.frame(NNinputs)
objectives <- data.frame(NNoutputs)
norm_object <- data.frame(scale(objectives))
weights <- data.frame(x$weights)
Training <- data.frame(parameters, norm_object)
input_colname <- colnames(parameters)
output_colname <- colnames(norm_object)
NNformula <- as.formula(c(paste(output_colname, '~', paste(input_colname, collapse =
"+"))))
ANN_0 <- neuralnet(NNformula, Training, hidden=x$nodes, threshold=c(0.1), rep=1,
algorithm='rprop+', startweights=weights, stepmax= 2e+05)
comp <- compute(ANN_0, data)
std <- sd(data.matrix(objectives))
mn <- mean(data.matrix(objectives))
predicton <- std*comp$net.result+mn
}
```

B.9 scatterplot Code

```
scatterplot <- function(table, plot=FALSE){
data <- table
Property <- data.frame(data$experimental, data$predicted)
experimental <- unique(Property[, 1])
expCount <- length(experimental)
predicted <- NULL
  id <- 1:expCount
  for (i in id){
  value <- Property[Property[,1]==experimental[i], ]
  Predicted <- mean(value[,2])
  predicted <- rbind(predicted, Predicted)
```

```
  }
x <- experimental
y <- predicted
names(x) <- 'experimental'
names(y) <- 'predicted'
if(plot==TRUE){
plot(x, y)
}
Data <- suppressWarnings(data.frame(x,y))
names(Data) <- c('experimental', 'predicted')
Data1 <- Data
}
```

B.10 svmcomp Code

```
svmcomp <- function(model, data){
x <- model
z <- svm(x=data.frame(SVinputs), y=data.frame(SVoutputs), scale=TRUE, type='eps-
regression', kernel=x$kernel, degree=x$dimensions ,coef0=2, fitted=TRUE,
epsilon=0.000001, shrinking=TRUE, cross=4,
probability=TRUE)
predictions <- predict(z, data)
data.frame(predictions)
}
```

B.11 svmOpt

```
svmOpt <- function(data=c('FlashPoint.csv'), fuelcomp= fuelcomp, inputCOLs=2:5,
outputCOL=6, kernels=c('radial', 'polynomial', 'linear', 'sigmoid'), simCOUNT=1000,
crossREF=1000, dimensionSim=1:2, ignorewarns=FALSE, debug=TRUE){
library(Metrics)
library(e1071)
if (ignorewarns==FALSE){
if (is.character(data)==FALSE){
print('The data files you put in to the function must exist as a list and as characters. Ex:
data=c(\'density.csv\', \'freezing.csv\').')
stop('data= is not as.character', call.=FALSE)
}
if (is.data.frame(inputCOLs==TRUE)){
print('For svmlOpt, you can only define the columns (as numbers) in which all of the
values lie. NOT a data.frame like svmSim.')
stop('inputCOLs is a data.frame', call.=FALSE)
}
if (length(outputCOL) != 1){
```

```
print('The functions in DataMine can not handle more more than one output at a time!
Just define them as separate datasets under data=')
stop('outputCOL is more than 1 column', call.=FALSE)
}
if (ncol(fuelcomp) != length(inputCOLs)){
print('The number of variables in fuelcomp= do not match the number of variables in
inputCOL=. You probably left the fuel blend percentages in the data.frame (from the
function fuelComp). Take it out of your data.frame!')
stop('fuelcomp= is not the same number of inputs as inputCOLs=', call.=FALSE)
}
}
if (debug==TRUE){
print('Debug forces 5 simulations and 5 cross references for each data file.')
print('Starting...')
}
datafiles <- length(data)
result <- matrix(data=NA, ncol=datafiles, nrow(fuelcomp))
id <- 1:length(data)
for (i in id){
if (debug==TRUE){print(paste('Initializing optimization of', data[i], '...'))}
Data_1 <- read.csv(data[i])
if (debug==TRUE){print(paste('Setting up data.frames for input and outputs for',
data[i]))}
SVinputs <- data.frame(Data_1[,inputCOLs])
inputs <- SVinputs
SVoutputs <- data.frame(Data_1[,outputCOL])
outputs <- SVoutputs
rows <- nrow(inputs)
if (debug==TRUE){
print(paste('Simulation start for neural network model of', data[i], '...'))
simCOUNT <- 5
crossREF <- 5
}
  hd <- 1:length(kernels)
  for (h in hd){
    jd <- 1:crossREF
    for (j in jd){
    sampling <- round(rows*.85)
    testingsize <- rows-sampling
    table <- matrix(data=NA, ncol=2, nrow=simCOUNT*testingsize)
    fd <- 1:simCOUNT
    for (f in fd){
    random <- sample(1:rows, sampling)
    parameters <- inputs[random, ]
```

```
    objectives <- outputs[random, ]
    dataplacement <- c(1:simCOUNT)-1
    q <- dataplacement[f]
    degree <- 3
    test_parameters <- inputs[-random, ]
    test_objectives <- outputs[-random, ]
    if (debug==TRUE){
    print(paste('[ Simulation Cross Reference', kernels[h], h, ']','Running function svm of',
data[i], '...', collapse=''))
    }
    svm0 <- svm(x=parameters, y=objectives, scale=TRUE, type='eps-regression',
kernel=kernels[h], degree=degree ,coef0=2, fitted=TRUE, epsilon=0.000001,
shrinking=TRUE, cross=4, probability=TRUE)
    if (debug==TRUE){
    print(paste('[ Simulation Cross Reference', kernels[h], ']','Model generated, predicting
values from', data[i], '...', collapse=''))
    }
    predictions <- predict(svm0, test_parameters)
    PvO <- data.frame(test_objectives, predictions)
    table[(q*testingsize+1):(f*testingsize),1] <- PvO[,1]
    table[(q*testingsize+1):(f*testingsize),2] <- PvO[,2]
    }
    }
    if (debug==TRUE){
    print(paste('[ Simulation Cross Reference', kernels[h], ']', 'SVM kernel calculation
complete for', data[i], collapse=''))
    print(paste('[ Simulation Data', kernels[h], ']', 'Kernel RMSE calculating for', data[i],
'...', collapse=''))
    }
    rmse <- table
    rootmean <- rmse(table[,1], table[,2])
    stats <- data.frame(degree, rootmean)
    colnames(stats) <- c('dimensions', kernels[h])
    if (h==1){
    best_stats <- stats
    }
    print(stats)
    if (debug==TRUE){
    print(paste('[ Simulation Data', kernels[h], ']', 'Comparing kernels against each other',
data[i], '...', collapse=''))
    print(stats)
    }
    if (stats[1,2] < best_stats[1,2]){
    best_stats <- stats
```

```
    }
    }
opt_kernel <- colnames(best_stats)[2]
print(paste(data[i], 'Optimal kernel found:', opt_kernel, 'with an RMSE of:',
best_stats[1,2]))
if (debug==TRUE){
print(paste('Best kernel established for', data[i], collapse=''))
print(opt_kernel)
}
incriments <- seq(8, (rows-4), 4)
  sample_table <- NULL
  yd <- 1:length(incriments)
  for (y in yd){
    ud <- 1:crossREF
    for (u in ud){
    dataset <- c(1:crossREF)-1
    q <- dataset[u]
    samplesize_0 <- incriments[y]
    testingsize <- rows-samplesize_0
    random_5 <- sample(1:rows, samplesize_0)
    parameters_5 <- inputs[random_5, ]
    objectives_5 <- outputs[random_5, ]
    test_parameters_5 <- inputs[-random_5,]
    test_objectives_5 <- outputs[-random_5,]
    svm5 <- svm(x=parameters_5, y=objectives_5, scale=TRUE, type='eps-regression',
kernel=opt_kernel, degree=degree ,coef0=2, fitted=TRUE, epsilon=0.000001,
shrinking=TRUE, cross=4, probability=TRUE)
    predictions_5 <- predict(svm5, test_parameters_5)
    sample_results <- data.frame(test_objectives_5, predictions_5)
    sample_table <- rbind(sample_table, sample_results)
    }
  rmse_PvO <- rmse(sample_table[,1], sample_table[,2])
  sample_rmse <- data.frame(samplesize_0, rmse_PvO)
  names(sample_rmse) <- c('samplesize', paste(data[i], 'RMSE'))
  print(sample_rmse)
  if (y==1){
  best_sample <- sample_rmse
  }
  if (sample_rmse[1,2] < best_sample[1,2]){
  best_sample <- sample_rmse
  }
  }
samplesize <- best_sample[1,1]
testingsize <- rows-samplesize
```

```
print(paste(data[i], 'Optimal samplesize found:', samplesize, 'with an RMSE of:',
print(best_sample[1,2])))
  gd <- 1:dimensionSim
  if (debug==TRUE){
  print(paste('[', opt_kernel, 'Dimension Optimization ]', 'Starting dimension loops for',
data[i], '...', collapse=''))
  gd <- 1:5
  dimensionSim <- 1:5
  }
  for (g in gd){
  degree_1 <- g
  xd <- 1:crossREF
  dataplacement <- c(xd)-1
  dim_bind <- matrix(data=NA, ncol=2, nrow=length(xd)*testingsize)
  if (debug==TRUE){
  print(paste('[ Dimension', g, 'Cross Reference ]', 'Starting dimension cross referencing
for', data[i], '...', collapse=''))
  }
    for (x in xd){
    q <- dataplacement[x]
    random_1 <- sample(1:rows, samplesize)
    parameters_1 <- inputs[random_1, ]
    objectives_1 <- outputs[random_1, ]
    test_parameters_1 <- inputs[-random_1, ]
    test_objectives_1 <- outputs[-random_1, ]
    if (debug==TRUE){
    print(paste('[ Dimension', g, 'Cross Reference ]', 'Running svm function for', data[i],
'...', collapse=''))
    }
    svm_1 <- svm(x=parameters_1, y=objectives_1, scale=TRUE, type='eps-regression',
kernel=opt_kernel, degree=degree_1 ,coef0=2, fitted=TRUE, epsilon=0.000001,
shrinking=TRUE, cross=4, probability=TRUE)
    if (debug==TRUE){
    print(paste('[ Dimension', g, 'Cross Reference ]', 'Predicting values for', data[i], '...',
collapse=''))
    }
    predictions_1 <- predict(svm_1, test_parameters_1) #predicting values
    table_1 <- data.frame(test_objectives_1, predictions_1)
    dim_bind[(testingsize*q+1):(x*testingsize),1] <- table_1[,1]
    dim_bind[(testingsize*q+1):(x*testingsize),2] <- table_1[,2]
    }
  if (debug==TRUE){
  print(paste('[ Dimension', g, 'Cross Reference ]', 'Calculating RMSE for', data[i], '...',
collapse=''))
```

```
  }
  rootmean_1 <- rmse(dim_bind[,1], dim_bind[,2])
  stats_1 <- data.frame(degree_1, rootmean_1)
  names(stats_1) <- c('dimensions', 'RMSE')
  if (g==1){
  best_dim <- stats_1
  }
  if (debug==TRUE){
  print(paste('[Dimension', g, ']', 'Comparing dimensions against each other for', data[i],
'...', collapse=''))
  print(stats_1)
  }
  if (stats_1[1,2] < best_dim[1,2]){
  best_dim <- stats_1
  }
  print(stats_1)
  opt_dim <- best_dim[1,1]
  }
print(paste(data[i], 'Optimal dimension found:', opt_dim, 'with an RMSE of:',
best_dim[1,2]))
opt_dim <- best_dim[1,1]
  sd <- 1:crossREF
  if (debug==TRUE){
  print(paste('[', data[i], 'Optimization, kernel',opt_kernel, 'Dimension', opt_dim, ']',
'Starting optimized svm calculations', collapse=''))
  sd <- 1:5
  }
  opt_svm <- matrix(data=NA, ncol=length(sd), nrow=nrow(fuelcomp))
  for (s in sd){
  random_2 <- sample(1:rows, samplesize)
  parameters_2 <- inputs[random_2, ]
  objectives_2 <- outputs[random_2, ]
  if (debug==TRUE){
  print(paste('[', data[i], 'Optimal Model Simuation Cross Reference', s, ']', 'Running
SVM...', collapse=''))
  }
  svm_2 <- svm(x=parameters_2, y=objectives_2, scale=TRUE, type='eps-regression',
kernel=opt_kernel, degree=opt_dim ,coef0=2, fitted=TRUE, epsilon=0.000001,
shrinking=TRUE, cross=4, probability=TRUE)
  if (debug==TRUE){
  print(paste('[', data[i], 'Optimal Model Simuation Cross Reference', s, ']', 'Predicting
input values...', collapse=''))
  }
  predictions <- predict(svm_2, fuelcomp)
```

```
  predicted <- data.frame(predictions)
  opt_svm[,s] <- predicted[,1]
  }
if (debug==TRUE){
print(paste('[', data[i], 'modeling complete', ']', 'Binding and averaging (from cross
referencing) predicted values for user input data.frame...', collapse=''))
}
averaged <- data.frame(rowSums(opt_svm))/ncol(opt_svm)
result[,i] <- averaged[,1]
if (debug==TRUE){
print(paste('[', data[i], 'results complete', '] Starting next simulation loop...'))
}
}
if (debug==TRUE){
print(paste('svmOpt complete'))
}
result <- data.frame(result)
names(result) <- data
result
}
```

B.12 svmSim Code

```
svmSim <- function(inputCOLs=2:5, outputCOL=6, kernel='polynomial',
simCOUNT=1000, dimensions=3, samplesize=4, ignorewarns=FALSE){
library(e1071)
if (ignorewarns==FALSE){
if (samplesize >= nrow(inputCOLs)){
print('You set your samplesize= higher than the amount of data you have.')
stop('samplesize too high', call.=FALSE)
}
if (is.data.frame(inputCOLs==FALSE)){
print('Your inputs and outputs must be in the form of data.frame(')
stop('inputCOLs not a data.frame', call.=FALSE)
}
if (length(outputCOL) > 1){
print('The functions in DataMine can not handle more more than one output at a time!')
stop('More than one column in outputCOL', call.=FALSE)
}
}
SVinputs <<- local(inputCOLs)
inputs <- SVinputs
SVoutputs <<- data.frame(outputCOL)
outputs <- SVoutputs
```

```
rows <- nrow(inputs)
id <- 1:simCOUNT
for (i in id){
random <- sample(1:rows, samplesize)
parameters <- inputs[random, ]
objectives <- outputs[random, ]
percenttesting <- (rows-samplesize)/rows
repeatsize <- 5.2/percenttesting
  table <- NULL
  fd <- 1:repeatsize
  for (f in fd){
  random <- sample(1:rows, samplesize)
  test_parameters <- inputs[-random, ]
  test_objectives <- outputs[-random, ]
  svm <- svm(x=parameters, y=objectives, scale=TRUE, type='eps-regression',
kernel=kernel, degree=dimensions ,coef0=2, fitted=TRUE, epsilon=0.000001,
shrinking=TRUE, cross=4, probability=TRUE)
  SV <- data.frame(svm$SV)
  predictions <- predict(svm, test_parameters)
  PvO <- data.frame(test_objectives, predictions)
  table <- rbind(table, PvO)
  }
data <- table
rmse <- table
rootmean <- rmse(data[,1], data[,2])
stats <- data.frame(0, rootmean)
colnames(stats) <- c('experimental', 'predicted')
colnames(data) <- c('experimental', 'predicted')
overall_file <- rbind(stats, data)
if (i==1){
overall <- overall_file
}
if (overall_file[1,2] < overall[1,2] & i != 1){
overall <- overall_file
}
}
data <- overall[-1,]
names(data) <- c('experimental', 'predicted')
rootmean <- overall[1,2]
names(rootmean) <- 'RMSE'
names(kernel) <- 'kernel'
names(dimensions) <- 'dimensions'
c(data, rootmean, SV, kernel, dimensions)
}
```

APPENDIX C

INTRODUCTION DATA AND FIGURES

Table 13 Volumes of global proven gas reserves, with respect to year.

Year	Trillion Cubic Meters	Year	Trillion Cubic Meters
1960	19062.3	1987	108364.7
1961	19930.5	1988	112226.9
1962	20878.7	1989	125815.4
1963	22928	1990	129839.4
1964	24058.1	1991	136674.7
1965	25024.5	1992	140011.2
1966	29705	1993	142365.8
1967	31594.1	1994	143527.8
1968	36952	1995	142675.2
1969	38059.8	1996	146566.1
1970	44960.7	1997	148874.3
1971	49857.4	1998	151720.9
1972	53915.9	1999	150618
1973	55640	2000	159665.2
1974	59970.2	2001	172846.8
1975	61648.7	2002	173441.9
1976	63374.2	2003	174974.3
1977	69263.3	2004	175256.7
1978	70714.1	2005	176156.6
1979	75268.7	2006	176333.1
1980	83978.4	2007	179446.5
1981	87791.3	2008	182112.3
1982	90426	2009	189082
1983	92722.6	2010	193385
1984	96059.1	2011	196657
1985	99580.9	2012	201079
1986	106446.9	2013	200363

Table 14 Data used in scatterplot of the work by Al-Nuami et. al. 2014.

Density	**(g/cm3)**	**Freezing Point**	**K**	**Flash Point**	**°C**	**Net Heat Content**	**MJ/kg**
Experimental	Predicted	Experimental	Predicted	Experimental	Predicted	Experimental	Predicted
0.7427	0.7428	217	207.1	328	321.8	0.2547	0.2547
0.7435	0.7434	218.6	210.2	327.7	321.1	0.2643	0.2649
0.7346	0.7346	215.9	218.8	327.7	321.1	0.2618	0.2618
0.7613	0.7613	227.1	221.4	327.5	320.5	0.2644	0.2654
0.7523	0.7521	222.4	233	328.2	325.2	0.267	0.2684
0.7447	0.7447	223.3	227.9	328.2	322.6	0.2692	0.2724
0.751	0.7508	242	228.5	328.5	322.4	0.2761	0.2785
0.7577	0.7575	230.2	230.2	328.2	324.3	0.2779	0.2804
0.7495	0.7496	230.5	232.7	327.5	323.6	0.2824	0.2851
0.765	0.7644	238.8	230.7	327.7	324.5	0.2879	0.2905
0.7715	0.771	224.8	232.9	329.6	327.8	0.2873	0.2899
0.7763	0.7804	221.6	233.3	329.8	326.2	0.2844	0.2868
0.7654	0.765	236	232.2	329.2	327.5	0.2958	0.2985
0.767	0.7663	232.1	235.3	328.5	326.4	0.2933	0.296
0.773	0.7724	245.5	242.1	328.8	326.9	0.2941	0.2964
0.7724	0.7717	246.6	242.9	328.7	327.7	0.2923	0.2946
0.77	0.7697	244.1	242.6	331.4	329.5	0.3044	0.3059
0.7765	0.7757	246.6	243.4	331.9	331.9	0.3045	0.306
0.7897	0.7889	243.2	236	332.4	332.2	0.3073	0.3083
0.7936	0.793	236.3	236.5	332.9	330	0.3071	0.3084
0.7949	0.7919	247.9	239.5	332.6	329.4	0.3036	0.3051
0.802	0.8012	239.5	239	332.2	329.8	0.3038	0.3051
0.8079	0.8067	243.2	243.2	330.2	330.1	0.3064	0.3071

Table 14 Continued

0.8273	0.8262	248	239.4	331.6	329.7	0.3057	0.3071
0.8091	0.8087	241.8	239.6	334.5	332.8	0.3089	0.3089
0.8301	0.8316	243.9	236.4	335	335	0.3109	0.3109
0.8281	0.8289			334	333	0.3084	0.3093
0.83	0.8222			334.2	334.1	0.309	0.3098
0.8461	0.8458			331.8	332.8	0.3084	0.3103
0.864	0.8633			332.6	331.2	0.304	0.3053
0.8531	0.8525			332.6	331.8	0.3017	0.3034
0.8569	0.8559			330.8	330	0.3006	0.3025
0.8628	0.8619			330.5	330.1	0.2989	0.301
0.8703	0.8693			330.2	330.9	0.2964	0.2988
0.8828	0.8828			330.3	328.8		

Table 15 Physical property analysis based on changes in aromatic composition.

Blend	Density (g/cm3)	Freezing Point (K)	Flash Point (°C)	Heat of Combustion MJ/kg
Peark SPK	0.755	-53	48.5	43.9
5% Sty.	0.76	-54	44	43.7
10% Sty.	0.768	-56	41.5	43.55
15% Sty.	0.775	-57	39	43.2

APPENDIX D

ARTIFICIAL NEURAL NETWORK DATA AND FIGURES

Table 16 Sample size root mean square errors.

Sample size	Density	Freezing Point	Flash Point	Heat Content
4	0.005633223	2.421579211	7.241465895	0.344486802
8	0.002625126	1.595705754	4.131269628	0.238808456
12	0.001611498	1.440084848	2.976467883	0.217547252
16	0.00118035	1.372211656		
20	0.000948633	1.331675078		
24	0.00082498	1.301241574		
28	0.000754426	1.265464899		
32	0.000712656	1.260974468		

Table 17 Global solver error analysis.

Technique	Function	RMSE
resilient backpropagation (with backtracking)	rprop+	3.454719
resilient backpropagation (without backtracking)	rprop-	3.769299
smallest learning rate	slr	3.735976
smallest absolute gradient	sag	3.684502

Table 18 Root mean square error of varying nodes of each physical property.

Nodes	Density	Freezing Point	Flash Point	Heat Content
1	0.0010613	0.77101677	0.09520232	2.051765803
2	0.0011321	0.759624676	0.090238982	2.039730185
3	0.0010989	0.787788166	0.089126335	1.948404595
4	0.0011476	0.719642289	0.099245183	1.85399423
5	0.001099	0.724199019	0.088638512	1.837844152
6	0.0011582	0.812333276	0.099523284	1.806711293
7	0.001303	0.787299362	0.099310913	1.771267895

Table 19 Scatterplot values.

Density	(g/cm3)	Freezing Point	K	Flash Point	°C	Net Heat Content	MJ/kg
Experimental	Predicted	Experimental	Predicted	Experimental	Predicted	Experimental	Predicted
0.7604	0.7617	236.8750	234.4655	51.9000	51.6308	45.4422	45.8227
0.7658	0.7667	237.1667	236.0782	57.4000	56.0522	47.1998	47.1264
0.7713	0.7725	236.8333	236.1610	55.8000	54.7571	46.2619	46.0875
0.7689	0.7704	227.7500	229.6712	47.1000	44.5076	46.4465	46.4436
0.7768	0.7791	233.7000	234.0697	46.4000	44.6781	46.7935	46.7968
0.7832	0.7857	238.1000	238.6640	41.5000	43.3447	47.0168	46.9840
0.7605	0.7613	236.5000	236.6769	35.3000	38.7440	46.8614	46.8004
0.7709	0.7715	227.3571	229.2401	58.3000	53.7886	46.0548	46.1643
0.7762	0.7774	228.5000	229.6475	59.7000	56.2631	46.4635	46.5645
0.7755	0.7767	229.8333	229.4415	48.2000	45.0232	46.2315	46.5375
0.7779	0.7796	227.3000	229.4334	41.0000	42.7273	46.4279	46.3232
0.7819	0.7835	238.5000	238.6985	57.8000	54.5114	46.5937	46.5666
0.7847	0.7861	241.3333	240.1843	56.9000	55.6503	46.7526	46.6922
0.7893	0.7902	235.0000	233.8976			46.3918	46.1691
0.7665	0.7669	228.6667	229.5873			46.4060	46.3741
0.7732	0.7734	241.5000	241.5702			46.0290	45.9694
0.7810	0.7822	231.3333	230.0047				
0.7802	0.7813	225.7500	229.8397				
0.7839	0.7858	234.6000	234.3590				
0.7865	0.7880	238.0000	238.9483				
0.7904	0.7917	228.3333	229.5672				
0.7932	0.7943	232.6250	231.9490				
0.7538	0.7563	235.1000	234.0276				

Table 19 Continued

0.7591	0.7607	231.6667	232.1650		
0.7732	0.7748	233.2500	234.0711		
0.7794	0.7815	237.6250	237.2549		
0.7644	0.7650	236.4000	236.9275		
0.7698	0.7707	229.8750	230.2154		
0.7707	0.7711	229.5000	229.7756		
0.7772	0.7778	236.8000	236.9741		
0.7646	0.7660	237.3750	238.3806		
0.7754	0.7770	242.0000	241.4321		
0.7746	0.7755	235.5000	236.0724		
0.7669	0.7673	240.5000	240.2505		

Table 20 Weights of optimal model used.

Density	Freezing Point	Flash Point	Heat Content
-0.1284	-0.1369	0.3889	-0.6941
-3.0582	2.8016	-0.1213	2.4830
-0.7838	-1.8099	1.0308	0.9894
9.7164	-6.1597	-0.6299	-1.8820
7.7878	-5.8779	2.6100	-1.6424
-2.9536	-0.9632	1.7019	-0.7814
5.2692	2.5046	-0.4166	-2.7576
	-2.3355	-1.4285	-2.0347
	-3.3370	0.3386	2.6128
	-5.7177	-1.0988	-2.5337
	1.1751	1.2915	2.9502
	8.2774	2.1626	0.8116
	-4.4183	2.5537	0.7189
	-7.7341	-0.4795	0.5671
	-5.9536	-0.9186	1.5726
	-0.1977	-0.1564	-0.6661
	3.3654	1.8681	-0.2257
	-2.8202	1.2449	-0.7686
	-3.7914	0.6499	2.2161
	-4.6056	-3.7227	-2.9058
	-2.4918	0.2956	0.2612
	0.5997	1.0148	-2.1219
	0.7861	1.9816	-1.5429
	2.2043	-1.4270	2.4038
	1.7494	-1.1237	1.5599
		-0.6347	1.9937
		-0.0812	1.3746
		-1.2314	-3.4386
		0.2185	0.1635
		2.3817	-0.5053
		1.2908	-1.0229

APPENDIX E

KRIGING INTERPOLATION DATA AND FIGURES

Table 21 Sample size root mean square errors of respective sample sizes.

Sample size	Density	Freezing Point	Flash Point	Heat Content
8	0.001299413	2.323647538	6.732320643	0.155481645
12	0.000624673	2.533970717	4.519383864	0.317191922
16	0.000563385	2.75880753		
20	0.000569995	1.780501082		
24	0.00047617	2.224411972		
28	0.00030038	1.599181887		
32	0.001268371	2.370056944		

Table 22 Optimal theta parameters generated and the respective errors.

	Density	Freezing Point	Flash Point	Heat Content
RMSD	0.600573939	0.129448135	0.014789151	0.148986451
θ_1	5	0.073226006	0.440672457	0.182094506
θ_2	0.228233473	5	5	2.246538848
θ_3	0.146952402	0.110048326	3.777486917	1.211477908
θ_4	0.00036496	1.589494758	2.936599662	0.107554408

Table 23 Scatter plot values of the various physiochemical properties.

Density	**(g/cm^3)**	**Freezing Point**	**K**	**Flash Point**	**°C**	**Heat Content**	**MJ/kg**
Experimental	Predicted	Experimental	Predicted	Experimental	Predicted	Experimental	Predicted
0.7538	0.7545	226.15	228.9557	57.4000	57.3924	46.5937	46.5559
0.7768	0.7771	227.3571	227.1243	56.9000	56.2329	46.7935	46.8696
0.7754	0.7748	228.5	227.4884	55.8000	56.9804	46.0290	46.0544
0.7669	0.7673	236.8	237.456	48.2000	46.0089	46.4060	46.4296
0.7810	0.7812	236.875	233.9956	59.7000	52.7313	46.7526	46.7118
0.7732	0.7727	240.5	240.8842	57.8000	56.0125	47.0168	46.9460
0.7779	0.7781	228.3333	228.9234	58.3000	55.0235	46.0548	46.0337
0.7819	0.7820	233.7	235.2982	47.1000	46.1006	45.4422	45.6695
0.7865	0.7862	235.1	234.6255	51.9000	55.8268	46.3918	46.3538
0.7591	0.7588	236.4	235.853	46.4000	46.3463	46.8614	46.7910
0.7713	0.7712	236.5	234.6259	41.0000	44.1917	46.4279	46.3510
0.7665	0.7667	228.6667	228.4899	41.5000	42.7151	46.2315	46.4671
0.7932	0.7935	229.875	233.5149			46.4465	46.3940
0.7689	0.7698	235	233.7646			46.4635	46.4524
0.7772	0.7771	242	242.6748			47.1998	47.0815
0.7802	0.7806	225.75	227.8251			46.2619	46.2078
0.7832	0.7837	227.3	228.3964				
0.7847	0.7847	233.25	232.7217				
0.7604	0.7595	236.8333	234.0304				
0.7605	0.7602	227.75	226.7681				
0.7709	0.7709	237.375	238.1412				
0.7707	0.7707	241.3333	240.2191				

Table 23 Continued

0.7762	0.7760	231.3333	227.6369		
0.7904	0.7903	235.5	238.1071		
0.7839	0.7838	232.625	232.2379		
0.7885	0.7900	234.6	236.1591		
0.7794	0.7795	237.625	237.3867		
0.7646	0.7649	238.1	238.1687		
0.7893	0.7889	238	238.614		
0.7644	0.7645	241.5	240.3272		
0.7755	0.7752	231.6667	230.9595		
0.7732	0.7732	237.1667	236.0129		
0.7658	0.7662	219.15	230.4647		
0.7746	0.7745	238.5	238.3231		
0.7698	0.7698	229.8333	229.5566		

APPENDIX F

SVM DATA AND FIGURES

Table 24 Density eps error plot comparisons based on kernel type.

Percent of Data	Linear	Polynomial	Radial	Sigmoid
0.001	0.97255	0.125340971	0.219432	0.0700628
0.002	1	0.24740862	0.438592	0.1580169
0.003	1	0.360883797	0.597298	0.251434
0.004	1	0.47040371	0.70565	0.3530456
0.005	1	0.575968358	0.760235	0.438678
0.006	1	0.672394981	0.807314	0.5278612
0.007	1	0.745362793	0.855076	0.5943731
0.008	1	0.807146754	0.896015	0.6576072
0.009	1	0.841653028	0.922626	0.710325
0.01	1	0.872067649	0.943914	0.7611308
0.011	1	0.896072013	0.954694	0.8086588
0.012	1	0.917075832	0.967795	0.8463535
0.013	1	0.929214403	0.976392	0.8784485
0.014	1	0.942034915	0.982123	0.9061732
0.015	1	0.951309329	0.984989	0.9336247
0.016	1	0.957037643	0.987718	0.9605299
0.017	1	0.961538462	0.990721	0.9748703
0.018	1	0.965630115	0.992495	0.9832013
0.019	1	0.970812875	0.995633	0.9896203
0.02	1	0.974222586	0.99768	0.9938541
0.021	1	0.976268412	0.998226	0.9971319
0.022	1	0.97913257	0.999318	1
0.023	1	0.981314785	1	1
0.024	1	0.982678669	1	1
0.025	1	0.984451718	1	1
0.026	1	0.98608838	1	1
0.027	1	0.987452264	1	1
0.028	1	0.988270595	1	1
0.029	1	0.989088925	1	1
0.03	1	0.990180033	1	1

Table 24 Continued.

0.031	1	0.991680306	1	1
0.032	1	0.992362248	1	1
0.033	1	0.992907801	1	1
0.034	1	0.993453355	1	1
0.035	1	0.99386252	1	1
0.036	1	0.994271686	1	1
0.037	1	0.994680851	1	1
0.038	1	0.995226405	1	1
0.039	1	0.995771959	1	1
0.04	1	0.996044735	1	1
0.041	1	0.996181124	1	1
0.042	1	0.996453901	1	1
0.043	1	0.997272231	1	1
0.044	1	0.997272231	1	1
0.045	1	0.99740862	1	1
0.046	1	0.997681397	1	1
0.047	1	0.997817785	1	1
0.048	1	0.997954173	1	1
0.049	1	0.997954173	1	1
0.05	1	0.997954173	1	1

Table 25 Density nu error plot comparisons based on kernel type.

Percent of Data	Linear	Polynomial	Radial	Sigmoid
0.001	0.97339	0.119268559	0.231168	0.0912937
0.002	1	0.238810044	0.441867	0.1775382
0.003	1	0.357805677	0.586381	0.26119
0.004	1	0.471206332	0.679995	0.3432041
0.005	1	0.577237991	0.745087	0.4137555
0.006	1	0.671533843	0.783433	0.4828057
0.007	1	0.749863537	0.818777	0.5406659
0.008	1	0.808406114	0.870497	0.5968886
0.009	1	0.846888646	0.908297	0.6516103
0.01	1	0.875409389	0.936681	0.7066048
0.011	1	0.895060044	0.955104	0.757369
0.012	1	0.91621179	0.966703	0.7968068

Table 25 Continued.

0.013	1	0.92849345	0.973526	0.8294214
0.014	1	0.940638646	0.978848	0.8606714
0.015	1	0.950054585	0.983761	0.8956059
0.016	1	0.957014192	0.986354	0.9264465
0.017	1	0.963837336	0.98881	0.9551037
0.018	1	0.969432314	0.991266	0.9710699
0.019	1	0.972161572	0.993859	0.9821234
0.02	1	0.974481441	0.99577	0.9866266
0.021	1	0.975982533	0.998362	0.9900382
0.022	1	0.978438865	1	0.9941321
0.023	1	0.979667031	1	0.9976801
0.024	1	0.981168122	1	1
0.025	1	0.982259825	1	1
0.026	1	0.984033843	1	1
0.027	1	0.985398472	1	1
0.028	1	0.986080786	1	1
0.029	1	0.987172489	1	1
0.03	1	0.988127729	1	1
0.031	1	0.989492358	1	1
0.032	1	0.990311135	1	1
0.033	1	0.99099345	1	1
0.034	1	0.99194869	1	1
0.035	1	0.992494541	1	1
0.036	1	0.993040393	1	1
0.037	1	0.993449782	1	1
0.038	1	0.994268559	1	1
0.039	1	0.994541485	1	1
0.04	1	0.99481441	1	1
0.041	1	0.99481441	1	1
0.042	1	0.995496725	1	1
0.043	1	0.995769651	1	1
0.044	1	0.995906114	1	1
0.045	1	0.996315502	1	1
0.046	1	0.996315502	1	1
0.047	1	0.996588428	1	1
0.048	1	0.996724891	1	1
0.049	1	0.996997817	1	1

Table 25 Continued.

0.05	1	0.996997817	1	1

Table 26 Freezing point eps error plot comparisons based on kernel type.

Percent of Data	Linear	Polynomial	Radial	Sigmoid
0.001	0.1641	0.060149864	0.121935	0.0600817
0.002	0.300532	0.123910082	0.233651	0.1235014
0.003	0.414679	0.185694823	0.332221	0.1856948
0.004	0.538981	0.24972752	0.420436	0.2449591
0.005	0.648494	0.31226158	0.503134	0.3091281
0.006	0.75644	0.37479564	0.576635	0.3717984
0.007	0.834946	0.43133515	0.653474	0.4302452
0.008	0.891781	0.490940054	0.70579	0.4871253
0.009	0.924765	0.542779292	0.746049	0.5425068
0.01	0.937168	0.588964578	0.780381	0.5980926
0.011	0.943914	0.63739782	0.813556	0.6439373
0.012	0.952842	0.683310627	0.842439	0.6854223
0.013	0.960883	0.726430518	0.866485	0.7190736
0.014	0.977648	0.761716621	0.889986	0.7558583
0.015	0.985689	0.784945504	0.908583	0.7891008
0.016	0.98971	0.806743869	0.927112	0.8214578
0.017	0.994139	0.823024523	0.943324	0.8551771
0.018	0.99632	0.838623978	0.955995	0.886921
0.019	0.997683	0.852520436	0.965191	0.9094005
0.02	0.999455	0.864986376	0.971526	0.9305858
0.021	1	0.879632153	0.977589	0.9448229
0.022	1	0.891008174	0.982221	0.955109
0.023	1	0.901430518	0.987057	0.9638965
0.024	1	0.909536785	0.989782	0.9711172
0.025	1	0.917438692	0.992439	0.976158
0.026	1	0.924386921	0.994619	0.9792234
0.027	1	0.930653951	0.997207	0.9822207
0.028	1	0.935626703	0.999251	0.9854905
0.029	1	0.940871935	1	0.9880109
0.03	1	0.944822888	1	0.9901226
0.031	1	0.949455041	1	0.9920981

Table 26 Continued.

0.032	1	0.953882834	1	0.9935967
0.033	1	0.957901907	1	0.9956403
0.034	1	0.961580381	1	0.997752
0.035	1	0.964918256	1	0.9990463
0.036	1	0.967983651	1	1
0.037	1	0.970844687	1	1
0.038	1	0.972479564	1	1
0.039	1	0.973910082	1	1
0.04	1	0.975408719	1	1
0.041	1	0.976771117	1	1
0.042	1	0.977861035	1	1
0.043	1	0.979700272	1	1
0.044	1	0.98119891	1	1
0.045	1	0.982833787	1	1
0.046	1	0.984332425	1	1
0.047	1	0.985694823	1	1
0.048	1	0.987057221	1	1
0.049	1	0.988147139	1	1
0.05	1	0.989032698	1	1

Table 27 Freezing point nu error plot comparisons based on kernel type.

Percent of Data	Linear	Polynomial	Radial	Sigmoid
0.001	0.15032	0.146302	0.117407	0.0349646
0.002	0.28838	0.292264	0.222283	0.0692475
0.003	0.419153	0.415971	0.316058	0.1054389
0.004	0.534328	0.52931	0.399551	0.1498092
0.005	0.649162	0.643897	0.481953	0.2040622
0.006	0.763248	0.763453	0.562994	0.2656761
0.007	0.845798	0.844828	0.63021	0.3212923
0.008	0.901308	0.891107	0.691773	0.3731598
0.009	0.927939	0.918194	0.730932	0.42012
0.01	0.939586	0.939564	0.766889	0.4620365
0.011	0.949394	0.94941	0.799101	0.5139722
0.012	0.959542	0.958666	0.831109	0.5736778
0.013	0.966898	0.966924	0.859507	0.6290213
0.014	0.971802	0.972028	0.88198	0.6806161

Table 27 Continued.

0.015	0.976434	0.972028	0.905067	0.7268266
0.016	0.981474	0.975839	0.921683	0.7572928
0.017	0.985492	0.980966	0.940888	0.7861232
0.018	0.990873	0.986479	0.954781	0.8225191
0.019	0.997071	0.992559	0.964315	0.8559842
0.02	1	0.997459	0.971942	0.8884269
0.021	1	1	0.977458	0.9185523
0.022	1	1	0.982362	0.9434297
0.023	1	1	0.985835	0.9604689
0.024	1	1	0.988287	0.9676254
0.025	1	1	0.991147	0.9721238
0.026	1	1	0.993258	0.9745774
0.027	1	1	0.99571	0.9781216
0.028	1	1	0.997753	0.9823473
0.029	1	1	0.999455	0.9873228
0.03	1	1	1	0.9914804
0.031	1	1	1	0.9951609
0.032	1	1	1	0.9986369
0.033	1	1	1	1
0.034	1	1	1	1
0.035	1	1	1	1
0.036	1	1	1	1
0.037	1	1	1	1
0.038	1	1	1	1
0.039	1	1	1	1
0.04	1	1	1	1
0.041	1	1	1	1
0.042	1	1	1	1
0.043	1	1	1	1
0.044	1	1	1	1
0.045	1	1	1	1
0.046	1	1	1	1
0.047	1	1	1	1
0.048	1	1	1	1
0.049	1	1	1	1
0.05	1	1	1	1

Table 28 Flash point eps error plot comparisons based on kernel type.

Percent of Data	Linear	Polynomial	Radial	Sigmoid
0.001	0.019235	0.012552	0.01096	0.0015972
0.002	0.031963	0.024705	0.033335	0.0053049
0.003	0.052568	0.036287	0.055083	0.0077006
0.004	0.075	0.053061	0.075404	0.0099823
0.005	0.099715	0.06413	0.102403	0.0138041
0.006	0.105479	0.082444	0.123295	0.0165421
0.007	0.137272	0.09608	0.142702	0.0193942
0.008	0.149429	0.117362	0.160854	0.0216188
0.009	0.158276	0.129857	0.177293	0.0235012
0.01	0.178368	0.142466	0.187168	0.0257829
0.011	0.189441	0.152907	0.202009	0.0287491
0.012	0.219578	0.16409	0.212569	0.0328561
0.013	0.229338	0.176699	0.227011	0.0351948
0.014	0.234589	0.190506	0.234831	0.0402145
0.015	0.25411	0.203058	0.242537	0.0451771
0.016	0.264041	0.214127	0.252811	0.0489989
0.017	0.276598	0.222172	0.261659	0.0525926
0.018	0.278482	0.232555	0.26828	0.0563573
0.019	0.279852	0.240657	0.276329	0.0593805
0.02	0.281107	0.250813	0.283635	0.064172
0.021	0.295377	0.261026	0.29117	0.0716445
0.022	0.3004	0.270554	0.299903	0.0744966
0.023	0.303025	0.281451	0.311034	0.079117
0.024	0.31387	0.290694	0.32245	0.0813416
0.025	0.327968	0.301991	0.33067	0.0840226
0.026	0.338071	0.316084	0.340602	0.0851064
0.027	0.342637	0.32584	0.350876	0.0872169
0.028	0.347945	0.33069	0.360237	0.0910387
0.029	0.352454	0.336795	0.372966	0.0970852
0.03	0.354509	0.341359	0.384725	0.0998232
0.031	0.374258	0.345809	0.394086	0.1031886
0.032	0.388071	0.352142	0.405845	0.1068393
0.033	0.393379	0.358019	0.41475	0.1086647
0.034	0.408105	0.363097	0.423654	0.1135702
0.035	0.423973	0.365322	0.432274	0.1205864
0.036	0.428311	0.366121	0.439123	0.1278307

Table 28 Continued.

0.037	0.430251	0.368118	0.444032	0.1335349
0.038	0.436416	0.370001	0.450083	0.1387827
0.039	0.438242	0.371883	0.45288	0.1438594
0.04	0.44218	0.374793	0.457047	0.1515031
0.041	0.447317	0.377475	0.464981	0.1585762
0.042	0.453938	0.379186	0.470175	0.1629114
0.043	0.456792	0.380898	0.477824	0.1698705
0.044	0.466781	0.383922	0.484445	0.1754036
0.045	0.473459	0.389228	0.490724	0.1820204
0.046	0.488756	0.392309	0.497974	0.1859563
0.047	0.495034	0.394135	0.506593	0.1893788
0.048	0.498744	0.396132	0.513157	0.1986196
0.049	0.499201	0.397957	0.520292	0.2023273
0.05	0.500856	0.40121	0.527313	0.2054076

Table 29 Flash point nu error plot comparisons based on kernel type.

Percent of Data	Linear	Polynomial	Radial	Sigmoid
0.001	0.004802	0.006097561	0.017935	0
0.002	0.023496	0.018292683	0.043352	0
0.003	0.042877	0.024390244	0.058602	0
0.004	0.05877	0.030487805	0.076765	0
0.005	0.095129	0.036585366	0.095271	0
0.006	0.107478	0.06097561	0.115947	0
0.007	0.121827	0.073170732	0.137252	0
0.008	0.135605	0.085365854	0.157128	0
0.009	0.146867	0.091463415	0.174549	0
0.01	0.158987	0.097560976	0.188999	0
0.011	0.166533	0.103658537	0.204764	0
0.012	0.187171	0.12195122	0.219557	0
0.013	0.195747	0.140243902	0.235035	0
0.014	0.204265	0.158536585	0.245431	0
0.015	0.21627	0.182926829	0.254798	0
0.016	0.223759	0.195121951	0.268563	0
0.017	0.235822	0.201219512	0.285355	0
0.018	0.244054	0.201219512	0.29358	0

Table 29 Continued.

0.019	0.248228	0.213414634	0.30249	0
0.02	0.253259	0.225609756	0.313342	0
0.021	0.261948	0.25	0.324252	0
0.022	0.275497	0.262195122	0.339331	0
0.023	0.2823	0.274390244	0.35361	0
0.024	0.287674	0.286585366	0.365661	0
0.025	0.292991	0.317073171	0.378798	0
0.026	0.30271	0.329268293	0.391307	0
0.027	0.313286	0.347560976	0.405986	0
0.028	0.323634	0.359756098	0.41541	0
0.029	0.334553	0.359756098	0.429804	0
0.03	0.345529	0.365853659	0.445454	0
0.031	0.347816	0.37195122	0.461789	0
0.032	0.35925	0.37195122	0.47127	0
0.033	0.367082	0.37195122	0.47881	0
0.034	0.373428	0.384146341	0.485321	0
0.035	0.379259	0.390243902	0.491832	0
0.036	0.38812	0.390243902	0.49663	0
0.037	0.400069	0.390243902	0.503141	0.0030818
0.038	0.408587	0.390243902	0.506626	0.0058213
0.039	0.413046	0.396341463	0.510738	0.0119279
0.04	0.42265	0.396341463	0.513651	0.0166648
0.041	0.433798	0.396341463	0.517763	0.0239699
0.042	0.4378	0.396341463	0.520676	0.0332154
0.043	0.447176	0.402439024	0.523703	0.0393791
0.044	0.452664	0.402439024	0.525588	0.043317
0.045	0.461182	0.402439024	0.527816	0.0495377
0.046	0.467299	0.408536585	0.530615	0.0536468
0.047	0.470386	0.408536585	0.534156	0.0597535
0.048	0.474274	0.408536585	0.535355	0.0650611
0.049	0.477761	0.408536585	0.537526	0.0723091
0.05	0.479133	0.414634146	0.543866	0.0795571

Table 30 Heat content eps error plot comparisons based on kernel type.

Percent of Data	Linear	Polynomial	Radial	Sigmoid
0.001	0.362598	0.225991	0.21449	0.1029622
0.002	0.544433	0.365405	0.463597	0.1636331
0.003	0.62152	0.490539	0.517844	0.3158458
0.004	0.720021	0.574259	0.607245	0.4254104
0.005	0.812812	0.627811	0.700928	0.5196288
0.006	0.883298	0.684577	0.733048	0.654354
0.007	0.895432	0.70814	0.828337	0.7837259
0.008	0.940043	0.723849	0.902926	0.8688437
0.009	0.940043	0.738843	0.93576	0.9081014
0.01	0.941113	0.752231	0.938258	0.9307637
0.011	0.960742	0.761514	0.941827	0.9403997
0.012	0.992505	0.771867	0.943255	0.9403997
0.013	1	0.783292	0.943255	0.9403997
0.014	1	0.79543	0.943255	0.9403997
0.015	1	0.808283	0.943255	0.9403997
0.016	1	0.81935	0.943255	0.9403997
0.017	1	0.823991	0.961813	0.9427195
0.018	1	0.829347	0.975196	0.9578872
0.019	1	0.833809	0.983405	0.9835832
0.02	1	0.838272	0.98965	1
0.021	1	0.842556	0.998751	1
0.022	1	0.84684	1	1
0.023	1	0.851303	1	1
0.024	1	0.857194	1	1
0.025	1	0.858979	1	1
0.026	1	0.859336	1	1
0.027	1	0.861121	1	1
0.028	1	0.864156	1	1
0.029	1	0.868975	1	1
0.03	1	0.870403	1	1
0.031	1	0.872189	1	1
0.032	1	0.874509	1	1
0.033	1	0.876473	1	1
0.034	1	0.877544	1	1
0.035	1	0.878079	1	1

Table 30 Continued.

0.036	1	0.878615	1	1
0.037	1	0.881114	1	1
0.038	1	0.883077	1	1
0.039	1	0.885041	1	1
0.04	1	0.886469	1	1
0.041	1	0.887719	1	1
0.042	1	0.88879	1	1
0.043	1	0.890039	1	1
0.044	1	0.891646	1	1
0.045	1	0.892538	1	1
0.046	1	0.893609	1	1
0.047	1	0.894323	1	1
0.048	1	0.89593	1	1
0.049	1	0.897001	1	1
0.05	1	0.898786	1	1

Table 31 Heat content nu error plot comparisons based on kernel type.

Percentage of Data	linear	poly	radial	sigmoud
0.001	0.286404	0.19697	0.202577	0.140307
0.002	0.466547	0.333333	0.373837	0.2625848
0.003	0.568157	0.454545	0.488547	0.3637986
0.004	0.683184	0.568182	0.613994	0.453588
0.005	0.80161	0.621212	0.723515	0.5662263
0.006	0.87746	0.659091	0.753042	0.6574438
0.007	0.885152	0.674242	0.810308	0.7425919
0.008	0.941682	0.681818	0.898533	0.86005
0.009	0.941682	0.689394	0.931102	0.9182435
0.01	0.941682	0.69697	0.93844	0.9425205
0.011	0.954025	0.704545	0.939513	0.9425205
0.012	1	0.727273	0.939692	0.9425205
0.013	1	0.742424	0.941661	0.9425205
0.014	1	0.742424	0.941661	0.9425205
0.015	1	0.757576	0.941661	0.9425205
0.016	1	0.772727	0.941661	0.9425205
0.017	1	0.772727	0.941661	0.9425205

Table 31 Continued.

0.018	1	0.780303	0.955082	0.9501964
0.019	1	0.780303	0.988726	0.9701892
0.02	1	0.780303	0.997674	0.9928597
0.021	1	0.780303	1	1
0.022	1	0.780303	1	1
0.023	1	0.787879	1	1
0.024	1	0.80303	1	1
0.025	1	0.810606	1	1
0.026	1	0.810606	1	1
0.027	1	0.810606	1	1
0.028	1	0.818182	1	1
0.029	1	0.825758	1	1
0.03	1	0.825758	1	1
0.031	1	0.825758	1	1
0.032	1	0.825758	1	1
0.033	1	0.825758	1	1
0.034	1	0.825758	1	1
0.035	1	0.833333	1	1
0.036	1	0.833333	1	1
0.037	1	0.833333	1	1
0.038	1	0.833333	1	1
0.039	1	0.833333	1	1
0.04	1	0.833333	1	1
0.041	1	0.833333	1	1
0.042	1	0.840909	1	1
0.043	1	0.840909	1	1
0.044	1	0.848485	1	1
0.045	1	0.848485	1	1
0.046	1	0.848485	1	1
0.047	1	0.848485	1	1
0.048	1	0.856061	1	1
0.049	1	0.863636	1	1
0.05	1	0.863636	1	1

Table 32 Density predicted vs. experimental values for each respective kernel.

Linear		**Polynomial**		**Radial**		**Sigmoidal**	
Experimental	Predicted	Experimental	Predicted	Experimental	Predicted	Experimental	Predicted
0.7538	0.7543	0.7646	0.7552	0.7646	0.7709	0.7538	0.7675
0.7604	0.7604	0.7658	0.7716	0.7713	0.7716	0.7646	0.7693
0.7658	0.7657	0.7732	0.7744	0.7689	0.7700	0.7604	0.7701
0.7713	0.7709	0.7768	0.7774	0.7732	0.7742	0.7713	0.7736
0.7732	0.7732	0.7754	0.7768	0.7768	0.7767	0.7768	0.7763
0.7768	0.7772	0.7794	0.7793	0.7754	0.7744	0.7754	0.7761
0.7754	0.7754	0.7832	0.7877	0.7698	0.7717	0.7832	0.7788
0.7794	0.7794	0.7644	0.7711	0.7669	0.7671	0.7644	0.7718
0.7832	0.7833	0.7669	0.7686	0.7709	0.7693	0.7698	0.7728
0.7698	0.7697	0.7709	0.7753	0.7762	0.7752	0.7755	0.7752
0.7709	0.7706	0.7762	0.7764	0.7755	0.7758	0.7819	0.7776
0.7762	0.7759	0.7755	0.7754	0.7779	0.7798	0.7847	0.7771
0.7779	0.7781	0.7779	0.7748	0.7819	0.7810	0.7707	0.7735
0.7819	0.7819	0.7819	0.7801	0.7893	0.7847	0.7732	0.7750
0.7893	0.7882	0.7893	0.7845	0.7707	0.7724	0.7810	0.7775
0.7732	0.7729	0.7665	0.7646	0.7746	0.7750	0.7802	0.7769
0.7772	0.7768	0.7732	0.7732	0.7772	0.7765	0.7839	0.7771
0.7839	0.7844	0.7865	0.7889	0.7810	0.7789	0.7865	0.7801
0.7904	0.7906	0.7904	0.7907	0.7802	0.7796	0.7904	0.7793
0.7932	0.7932	0.7932	0.8000	0.7865	0.7823	0.7932	0.7811
0.7591	0.7596	0.7538	0.7403	0.7932	0.7853	0.7591	0.7700
0.7847	0.7843	0.7591	0.7600	0.7591	0.7657	0.7658	0.7722
0.7665	0.7668	0.7604	0.7542	0.7658	0.7665	0.7689	0.7735

Table 32 Continued.

0.7707	0.7707	0.7605	0.7601	0.7832	0.7793	0.7732	0.7746
0.7810	0.7808	0.7698	0.7699	0.7605	0.7645	0.7794	0.7760
0.7802	0.7804	0.7839	0.7794	0.7644	0.7661	0.7669	0.7730
0.7865	0.7868	0.7847	0.7788	0.7732	0.7732	0.7779	0.7752
0.7646	0.7648	0.7746	0.7742	0.7604	0.7658	0.7893	0.7796
0.7605	0.7605	0.7772	0.7758	0.7794	0.7781	0.7772	0.7748
0.7644	0.7645	0.7810	0.7812	0.7847	0.7836	0.7605	0.7701
0.7669	0.7666	0.7802	0.7789	0.7839	0.7849	0.7709	0.7737
0.7755	0.7755	0.7713	0.7707	0.7538	0.7660	0.7762	0.7755
0.7746	0.7747	0.7707	0.7717	0.7665	0.7702	0.7665	0.7717
0.7689	0.7693	0.7689	0.7718	0.7904	0.7853	0.7746	0.7755

Table 33 Freezing point predicted vs. experimental values for each respective kernel.

Linear		**Polynomial**		**Radial**		**Sigmoidal**	
Experimental	Predicted	Experimental	Predicted	Experimental	Predicted	Experimental	Predicted
235.5000	235.9749	241.5000	244.0383	229.8750	232.2033	242.0000	238.7155
231.3333	229.2369	236.8333	235.4635	235.5000	236.5882	235.5000	234.8986
238.0000	238.9671	242.0000	245.3460	231.3333	229.7183	231.3333	232.4978
233.2500	233.8048	235.5000	236.3740	236.8750	233.2488	238.0000	236.5592
228.5000	229.0471	228.5000	226.3549	238.5000	236.3618	236.8750	234.1844
240.5000	240.7631	241.3333	240.1688	228.5000	231.4064	225.7500	232.2417
237.1667	235.7392	227.7500	230.9732	240.5000	239.3241	240.5000	237.2571
229.5000	229.3572	237.6250	235.5816	229.5000	230.3726	241.3333	237.4688
241.3333	240.2634	228.3333	231.6431	241.3333	239.2624	236.8333	234.9422

Table 31 Continued.

237.6250	236.9033	236.8000	238.1896	236.8333	236.0768	227.7500	232.3772
233.7000	233.7362	235.0000	234.3324	237.6250	237.7925	237.6250	235.6424
228.3333	228.9178	238.1000	239.6980	233.7000	234.2849	233.7000	234.3146
236.8000	236.6748	228.6667	227.9592	228.3333	228.5380	228.3333	232.1688
229.8333	228.3977	237.3750	239.1364	236.8000	236.7803	238.1000	236.7327
238.1000	238.9824	235.1000	234.5424	238.1000	237.2829	228.6667	231.6297
228.6667	229.0779	236.4000	236.5924	228.6667	230.3542	235.1000	234.1764
236.4000	236.7595	231.6667	233.8207	235.1000	233.8248	236.4000	236.2942
227.3000	228.7152	227.3000	228.6494	236.5000	235.3961	227.3000	231.8060
236.5000	236.3884	236.5000	239.2203	232.6250	232.6377	236.5000	237.2881
232.6250	231.4002	232.6250	231.5021	227.3571	230.8339	232.6250	233.5647
236.8333	235.7209	229.8750	229.1565	242.0000	238.5284	241.5000	238.6563
242.0000	242.4340	231.3333	224.9658	238.0000	238.6754	229.8750	232.3338
236.8750	233.8800	238.0000	237.8647	225.7500	230.8284	229.8333	231.7735
238.5000	238.5892	225.7500	229.8298	233.2500	234.5597	237.3750	237.0494
227.7500	229.2269	233.2500	236.0904	237.1667	236.2609	231.6667	233.6077
234.6000	233.8039	240.5000	241.4100	235.0000	233.8989	227.3571	231.4480
237.3750	238.6742	237.1667	235.9149	229.8333	229.5391	238.5000	237.4478
235.1000	233.5849	234.6000	236.1997	231.6667	232.1163	233.2500	234.4177
229.8750	229.6317	227.3571	225.5811	227.3000	229.5208	228.5000	231.7138
225.7500	229.2817	236.8750	234.3569	227.7500	229.1585	229.5000	232.1745
235.0000	233.3482	229.8333	229.2112	234.6000	234.3673	236.8000	236.3253
227.3571	228.5467	233.7000	235.7287	237.3750	237.6298	235.0000	234.2283
241.5000	242.8391	238.5000	239.9865	236.4000	236.4288	234.6000	234.2168
231.6667	231.8606	229.5000	227.7150	241.5000	238.2332	237.1667	234.8270

Table 34 Flash point predicted vs. experimental values for each respective kernel.

Linear		**Polynomial**		**Radial**		**Sigmoidal**	
Experimental	Predicted	Experimental	Predicted	Experimental	Predicted	Experimental	Predicted
51.9000	58.7450	56.9000	56.5335	51.9000	56.2421	51.9000	60.6430
57.4000	56.6135	57.8000	54.8768	59.7000	54.3501	57.4000	53.0605
58.3000	55.9087	58.3000	73.2726	57.4000	56.8598	55.8000	54.0602
48.2000	51.8403	47.1000	37.2514	48.2000	50.0879	47.1000	52.8833
46.4000	38.1451	46.4000	5.5563	41.0000	48.5907	46.4000	50.7621
59.7000	55.5510	59.7000	56.3681	55.8000	57.1441	59.7000	52.7696
56.9000	56.8875	57.4000	57.3387	57.8000	55.5109	48.2000	54.4744
47.1000	44.9613	55.8000	57.0370	46.4000	48.3633	41.0000	53.6877
41.0000	49.6024	41.0000	52.1495	58.3000	53.6257	57.8000	54.1536
55.8000	57.2945	51.9000	98.6230	41.5000	47.2178	58.3000	55.6412
41.5000	46.3070	41.5000	50.3255	56.9000	56.9455	41.5000	52.4309
57.8000	56.5389	48.2000	32.5083	47.1000	44.3845	56.9000	53.5389

Table 35 Heat content predicted vs. experimental values for each respective kernel.

Linear		**Polynomial**		**Radial**		**Sigmoidal**	
Experimental	Predicted	Experimental	Predicted	Experimental	Predicted	Experimental	Predicted
45.4422	45.9545	45.4422	43.8660	46.7526	46.6806	46.4279	46.5460
47.1998	47.2385	46.4465	46.4759	46.0290	46.2164	46.0290	46.2130
46.7935	46.7985	46.4060	46.3685	46.8614	46.5003	46.8614	46.5315
46.8614	46.6097	46.4635	46.4662	46.4635	46.6367	46.4635	46.4775
47.0168	47.0042	46.2315	46.4319	45.4422	46.2648	46.2315	46.5849

Table 35 Continued

46.0290	46.0575	47.1998	59.5457	47.1998	46.8309	46.4465	46.5689
46.2619	46.1209	46.4279	46.3386	46.7935	46.8255	46.5937	46.5205
46.5937	46.5746	46.3918	46.1675	46.3918	46.1857	46.7526	46.5215
46.3918	46.1877	46.2619	46.7197	46.5937	46.5713	46.2619	46.4843
46.2315	46.5694	46.5937	46.5723	47.0168	46.9496	47.1998	46.8635
46.7526	46.6847	46.7935	46.9007	46.2619	46.2152	46.0548	46.3456
46.0548	46.2413	46.8614	46.7082	46.4465	46.5051	46.7935	46.5455
46.4635	46.6341	46.7526	46.7452	46.4060	46.3714	47.0168	46.6427
46.4060	46.3745	47.0168	45.7088	46.0548	46.2687	46.3918	46.2328
46.4465	46.4863	46.0290	46.0462	46.2315	46.5444	45.4422	46.2791
46.4279	46.3498	46.0548	46.1955	46.4279	46.3763	46.4060	46.3553

APPENDIX G

PHYSICAL PROPERTY TRAINING DATA

Table 36 Freezing point raw data.

n-Decane	Sol T	Decalin	Toluene	Experimental Temperature (K)
0.65	0.25	0	0.1	241.500
0.65	0.2	0.05	0.1	242.000
0.6	0.25	0	0.15	240.500
0.6	0.2	0.05	0.15	241.333
0.55	0.25	0.1	0.1	238.000
0.55	0.25	0	0.2	238.100
0.55	0.2	0.15	0.1	238.500
0.55	0.2	0.05	0.2	237.375
0.5	0.25	0.1	0.15	237.625
0.5	0.2	0.15	0.15	236.800
0.5	0.2	0.1	0.2	236.400
0.5	0.15	0.15	0.2	236.500
0.45	0.45	0	0.1	236.833
0.45	0.4	0.05	0.1	235.500
0.45	0.4	0	0.15	237.167
0.45	0.35	0.05	0.15	236.833
0.4	0.4	0.1	0.1	236.875
0.4	0.4	0	0.2	234.600
0.4	0.35	0.15	0.1	233.250
0.4	0.35	0.1	0.15	233.700
0.4	0.35	0.05	0.2	235.100
0.4	0.3	0.15	0.15	235.000
0.35	0.35	0.1	0.2	231.667
0.35	0.3	0.15	0.2	232.625
0.25	0.65	0	0.1	229.875
0.25	0.6	0.05	0.1	231.333
0.25	0.6	0	0.15	229.500
0.25	0.55	0.1	0.1	225.750
0.25	0.55	0.05	0.15	227.750
0.25	0.55	0	0.2	228.667

Table 36 Continued.

0.25	0.5	0.15	0.1	228.500
0.25	0.5	0.1	0.15	228.333
0.25	0.5	0.05	0.2	228.667
0.25	0.45	0.15	0.15	229.833
0.25	0.45	0.1	0.2	227.300
0.25	0.4	0.15	0.2	227.357

Table 37 Density raw data.

n-Decane	Sol T	Decalin	Toluene	Experimental Density (g/cm^3)
0.65	0.25	0	0.1	0.754
0.45	0.45	0	0.1	0.759
0.25	0.65	0	0.1	0.765
0.65	0.2	0.05	0.1	0.760
0.45	0.4	0.05	0.1	0.766
0.25	0.6	0.05	0.1	0.771
0.55	0.25	0.1	0.1	0.769
0.4	0.4	0.1	0.1	0.773
0.25	0.55	0.1	0.1	0.777
0.55	0.2	0.15	0.1	0.775
0.4	0.35	0.15	0.1	0.779
0.25	0.5	0.15	0.1	0.783
0.6	0.25	0	0.15	0.761
0.45	0.4	0	0.15	0.764
0.25	0.6	0	0.15	0.770
0.6	0.2	0.05	0.15	0.767
0.45	0.35	0.05	0.15	0.771
0.25	0.55	0.05	0.15	0.776
0.5	0.25	0.1	0.15	0.776
0.4	0.35	0.1	0.15	0.778
0.25	0.5	0.1	0.15	0.782
0.5	0.2	0.15	0.15	0.782
0.4	0.3	0.15	0.15	0.785
0.25	0.45	0.15	0.15	0.789
0.55	0.25	0	0.2	0.766
0.4	0.4	0	0.2	0.771

Table 37 Continued.

0.25	0.55	0	0.2	0.775
0.55	0.2	0.05	0.2	0.773
0.4	0.35	0.05	0.2	0.777
0.25	0.5	0.05	0.2	0.781
0.5	0.2	0.1	0.2	0.780
0.35	0.35	0.1	0.2	0.784
0.25	0.45	0.1	0.2	0.787
0.5	0.15	0.15	0.2	0.787
0.35	0.3	0.15	0.2	0.790
0.25	0.4	0.15	0.2	0.793

Table 38 Flash point raw data.

n-Decane	Sol T	Decalin	Toluene	Experimental Flash Point (°C)
0.3555	0.4345	0.2	0.01	51.9
0.135	0.165	0.7	0	59.7
0.18	0.22	0.6	0	57.4
0.2025	0.2475	0.55	0	57.4
0.225	0.275	0.5	0	56.9
0.2475	0.3025	0.45	0	55.8
0.27	0.33	0.4	0	57.8
0.315	0.385	0.3	0	58.3
0.0135	0.0165	0.7	0.27	48.2
0.0135	0.0165	0.5	0.47	47.1
0.0135	0.0165	0.3	0.67	46.4
0.0225	0.0275	0.6	0.35	41
0.0225	0.0275	0.4	0.55	41.5

Table 39 Heat content raw data.

n-Decane	Sol T	Decalin	Toluene	Experimental Heat Content(MJ/kg)
0.054	0.066	0.8	0.08	45.442
0.205	0.2503	0.5	0.045	46.232
0.356	0.4345	0.2	0.01	47.200
0.135	0.165	0.7	0	46.262
0.180	0.22	0.6	0	46.428
0.203	0.2475	0.55	0	46.447
0.225	0.275	0.5	0	46.594
0.248	0.3025	0.45	0	46.753
0.270	0.33	0.4	0	46.794
0.315	0.385	0.3	0	47.017
0.005	0.0055	0.6	0.39	46.392
0.014	0.0165	0.7	0.27	46.029
0.014	0.0165	0.5	0.47	46.406
0.014	0.0165	0.3	0.67	46.861
0.023	0.0275	0.6	0.35	46.055
0.023	0.0275	0.4	0.55	46.464